Solving Queueing Design and Control Problems with Constraint Programming

by

Daria Terekhov

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Mechanical and Industrial Engineering
University of Toronto

Abstract

Solving Queueing Design and Control Problems with Constraint Programming

Daria Terekhov

Master of Applied Science

Graduate Department of Mechanical and Industrial Engineering

University of Toronto

2007

The central thesis of this dissertation is that constraint programming can be effective in solving queueing design and control optimization problems. We consider a retail facility that has front room and back room operations, and cross-trained workers. Firstly, we examine the queueing control problem of optimizing a policy for switching workers between the two rooms so that expected waiting time in the front room queue is minimized and all back room work is performed. Secondly, we address the queueing design and control problem of finding a smallest-cost combination of cross-trained and specialized workers that guarantees the existence of a satisfactory control policy. We propose several constraint programming methods for these problems and experimentally demonstrate their strong performance. We then analyze our second problem under a complete set of assumptions regarding worker costs. To our knowledge, this is the first work that integrates the methodologies of constraint programming and queueing theory.

Acknowledgements

There are several people that I would like to thank for helping me to write this dissertation.

I would firstly like to thank my supervisor, Professor Chris Beck, for being such a great teacher, advisor and role model, for all of your ideas, for your instantaneous e-mail replies and for taking the time to explain all aspects of research to me. Thank you also for the amazing opportunity to go to Ireland, where I have definitely learned a lot, and for the opportunities to go to CPAIOR and AAAI.

I would also like to thank Ken Brown, for suggesting many of the ideas that are the basis of Chapter 4 (and the AAAI paper), for supervising my work while I was in Ireland, for proof-reading Chapter 5, and for all your help.

Thanks to Tom Carchrae, for all of the very useful discussions about research, experiments and thesis-writing, and for becoming a great friend to me in such a short amount of time.

Thanks to all of the people that have discussed the "back room/front room" problem with me (most of whom I met at 4C). In particular, thanks to Armagan Tarim for making it clear to me that the term "optimal policy" has two different interpretations.

Thanks to all the people in my lab – you have all helped me at some point. In particular, I need to say thank you to Ivan Heckman for answering all my questions about Linux, ILOG and the cluster, and to Anna Rogóz, for the many helpful discussions and your help with the queueing design and control part of the literature review chapter.

A big thank you to my Best Friend – for ensuring that I always have something to read other than papers, something to listen to other than lectures, something to write other than my dissertation and something to decode other than my computer programs; for all your valuable advice, your continuous support, and for being a most excellent distraction from my work!

I would also like to thank my Grand-Parents for always inspiring and supporting me.

Most importantly, a THANK YOU to my Parents, for all your love, support and advice, for inspiring me, for treating me like royalty every single day, for teaching me to work hard, for listening to my convoluted explanations of what I do, for believing in me, and for always knowing that the "back room/front room problem" is the most important problem in the world!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The central thesis of this dissertation is that constraint programming can be effective in solving queueing design and control optimization problems. We demonstrate this thesis by

- presenting a complete constraint programming method for solving a queueing control problem for which only a heuristic method existed previously,

- developing a complete constraint programming approach for a queueing design and control problem which is a generalization of a problem that only has a heuristic solution in the literature.

To our knowledge, this is the first work which combines the models and methods of constraint programming and queueing theory. Therefore, it both broadens the scope of problems that can be solved using constraint programming and extends the range of methods available for solving queueing-based optimization problems.

1.1 Motivations

Constraint programming (CP) was originally developed for solving problems in artificial intelligence and computer science, such as machine vision [54]. It has also proven to be a very

successful approach for many problems in operations research (OR), such as scheduling, routing and inventory planning [9, 27, 75, 92]. The success of constraint programming in solving a variety of problems demonstrates its flexibility and provides the motivation to apply it to another area of operations research–optimal design and control of queueing systems. In more detail, the motivations for the work presented in this dissertation are:

1. **The Need for Effective Methods for Solving Stochastic OR Problems**

   All real-world problems are affected by uncertainty and change. Therefore, it is often important to take these two aspects of a problem into account when modelling and solving it mathematically. However, the creation of a model and a resolution technique is usually significantly more difficult for a stochastic problem than for its deterministic counterpart. In this dissertation, we examine a new direction in the development of methods for stochastic problems: the integration of constraint programming and queueing theory.

2. **The Application of Constraint Programming to Stochastic Resource Allocation**

   Deterministic resource allocation problems have long been studied by researchers in constraint programming [6, 36]. More recently, there has also been work in the artificial intelligence community on developing methods for stochastic versions of such problems [19]. Resource allocation under uncertainty is also of interest in operations research. For example, in problems dealing with the optimal design and control of queueing systems, it is often necessary to determine how to assign servers to different types of jobs or how many servers to employ so as to optimize some measure of system performance [42, 87]. In order to extend the variety of resource allocation problems that could be addressed by constraint programming, we have decided to test the usefulness of CP for queueing design and control problems.

3. **The Integration of Constraint Programming and Queueing Theory**

   To our knowledge, there has been no previous work on the integration of methodologies from constraint programming and queueing theory. However, the existence of a

connection between the two may have interesting implications in both research fields: it will broaden the scope of problems that can be addressed by constraint programming and will extend the range of methods available for solving queueing-based problems. Consequently, this dissertation examines the applicability of constraint programming to a particular sub-area of queueing theory–the optimal design and control of queues.

4. **The Application of Constraint Programming to Operations Research Problems** Although CP has been successfully applied to many operations research problems [9, 27, 75, 92], it is still not as well-known or as widely used in the operations research community as traditional mathematical programming techniques for optimization problems. In particular, it has not been used to solve numerical optimization problems arising in queueing theory. Thus, by addressing two queueing design and control problems with constraint programming, we intend to promote the use of CP for queueing-based optimization problems.

1.2 Outline

The outline of the dissertation is as follows:

Chapter 2 provides the necessary background for the work presented in this dissertation. We start by discussing constraint programming and the extensions of constraint programming that have been developed for solving problems involving uncertainty and change. Subsequently, an overview of material on queueing control and design problems is presented. Further, we survey the literature on the subject of cross-training, since problems addressed in the following chapters deal with a retail facility which employs cross-trained workers. We conclude the chapter with a detailed discussion of the two problems that are of interest in this dissertation. Both problems are based on the work of Berman et al. [15], with the first being a queueing control problem, and the second involving elements of both queue design and queue control.

In Chapter 3, we propose three constraint programming models for a queueing control

problem in which the goal is to optimize a particular type of policy for switching cross-trained workers between two job types in a retail facility. Further, we present two specialized constraint programming inference methods for improving the efficiency of the CP models: dominance rules and shaving. Experimental results show that the best constraint programming model, using shaving, is able to find optimal solutions and prove optimality for many problem instances within a reasonable run-time. However, an existing heuristic algorithm performs better in terms of solution quality over time. A hybrid method combining the heuristic and the best constraint programming method is shown to perform better than either of these approaches separately. We conclude the chapter with a discussion of the differences between the three CP models and an analysis of their inability to prove optimality in some cases.

In Chapter 4, we consider a retail facility with front room and back room operations which has the option of hiring specialized or cross-trained workers. Under specific assumptions regarding the staffing costs of the different types of employees, we seek a smallest-cost combination of cross-trained and specialized workers that satisfies constraints on the expected customer waiting time and expected number of workers in the back room. This problem is an extension of the problem considered in Chapter 3. A logic-based Benders' decomposition method is proposed. In this method, both the master problem of identifying the lowest-cost staffing configuration and the sub-problem of finding a feasible control policy are modelled and solved using constraint programming. Experimental results demonstrate the strong performance of this approach across a wide variety of problem parameters. An analysis of the contributions of the main components of this method to its overall performance is presented.

In Chapter 5, we examine the structure of the optimal solution to the staff mix problem of the preceding chapter under various staffing cost assumptions. In particular, we prove that, in four of the five cost cases, the optimal solution can be easily identified, while in the remaining case, a more complex approach is required for solving the problem. As part of this examination, we identify a weakness in the control policy formulation proposed in previous work and suggest a new policy formulation. We discuss the implications of the policy definition on the structure

of the optimal solution and show that the techniques and optimal solutions for the problem under the cost assumptions of the previous chapter remain valid with the new policy definition.

Chapter 6 concludes this dissertation by re-stating its main contributions and suggesting some areas for future work.

A list of symbols is included after Chapter 6, stating the numbers of the pages on which the first definition of a symbol can be found. Only the symbols that are essential for the definition of the presented problems or models are included. If the same symbol is used in different chapters with a slightly different interpretation, the page numbers provided point to the first page mentioning each definition.

1.3 Summary of Contributions

The contributions of the dissertation are as follows:

- Constraint programming is shown to be useful for solving a queueing control optimization problem.

- Several complete methods are proposed for a problem for which only a heuristic method existed previously. The best constraint programming method is able to prove optimality or find the best-known solution in many problem instances. A hybrid approach, based on a combination of the best constraint programming model and the existing heuristic, is shown to find good-quality solutions in a substantially shorter amount of time than the best pure CP model and to be just as effective in proving optimality.

- We demonstrate that constraint programming can be used to solve a problem which deals with both optimal queue design and optimal queue control.

- We propose a realistic new problem based on existing work. Furthermore, we provide a complete constraint programming method for solving it. This method performs well

over a wide range of problem parameters, being able to find the optimal solution for most instances within a reasonable run-time.

- We provide proofs of the structure of the optimal solution to the queueing design and control problem of interest under various assumptions. Additionally, we identify a weakness in the control policy formulation proposed in previous work and suggest a new policy formulation.

- By demonstrating that constraint programming can be a useful approach for queueing design and control problems, we make the first step towards the integration of constraint programming and queueing theory.

Chapter 2

Literature Review

This chapter starts with a survey of three research areas which may, at first glance, appear unrelated: constraint programming, an alternative to the traditional operations research methods for solving optimization problems; optimal design and control of queues, a class of optimization problems based on queueing theory; and the area dealing with issues related to cross-training of workers. Following this survey, two problems are presented which are the main focus of this dissertation. These problems deal with the optimal control and design of a queueing system in a retail facility employing cross-trained workers, and will be used, in the following chapters, to investigate the applicability of constraint programming for solving such queueing theory-based optimization problems.

2.1 Constraint Programming

Although originally developed for solving problems in artificial intelligence and computer science, constraint programming (CP) has been proven to be a successful approach for many problems in operations research, such as scheduling, routing and inventory planning [9, 27, 75, 92]. CP is both an expressive modelling language that does not require the user to know all of the intricate details of how the problem is going to be solved and a study of algorithms that are flexible enough to support such a modelling language.

Problems that are of interest in constraint programming are called constraint satisfaction problems (CSPs). A CSP is defined as a set of variables together with a set of domains which contain values that are allowed to be assigned to these variables, and a set of constraints which restrict the combinations of values that the variables can take on. More formally, a CSP is a triple $< X, D, C >$, where X is an n-tuple of variables $< x_1, x_2, \ldots, x_n >$, D is an n-tuple of domains $< D_1, D_2, \ldots, D_n >$, such that the values that can be assigned to the variable x_i come from domain D_i, and C is a set of t constraints $< C_1, C_2, \ldots, C_t >$ [37]. Each $C_j \in C$ is a pair $< \sigma, \rho >$, where σ is the list of variables involved in this constraint, called the scope, and ρ is the subset of the Cartesian product of their domains which defines the tuples that are allowed by this constraint, called the constraint relation [82]. The relation can be represented either extensionally, by explicitly listing all of the satisfying tuples, or intensionally, by providing a mathematical expression that restricts the values that can be assigned to the variables in σ [82]. A solution to a CSP is an assignment to each variable x_i of a value from its domain, D_i, which satisfies all of the constraints in C.

CP also deals with constraint satisfaction optimization problems (CSOPs), in which the goal is to find a solution satisfying all constraints of the problem and minimizing or maximizing some objective function [8]. Thus, a CSOP can be formally defined as a CSP ($< X, D, C >$) that is combined with an objective function, $f(s)$, where s is some subset of the variables in X which maps every solution of the CSP to a numerical value [8, 82].

In the following sections, we discuss the general methods which are used for solving and modelling CSPs and CSOPs. Further on, we discuss how CP has been extended for dealing with problems involving uncertainty and change.

2.1.1 General Approach for Solving CSPs

CSPs and CSOPs are usually solved using complete methods, which are based on combining systematic search with constraint propagation, or incomplete methods, such as local search [72]. As all constraint programming models proposed in this dissertation are solved using a

combination of systematic search and constraint propagation, this method is the focus of this section. The reader is referred to Chapter 5 of the *Handbook of Constraint Programming* [50] for a discussion of local search methods used in CP.

2.1.1.1 Constraint Propagation

Constraint propagation is the term used to describe the process of reducing variable domains by making inferences from the constraints of the problem. Another term used to describe the action of reducing the domains of variables due to propagation is "pruning". In a constraint programming solver, each type of constraint usually has an associated propagation algorithm. The propagation algorithm aims to achieve a pre-specified level of consistency, that is, a state in which it will not be able to make any more domain reductions [82].

A detailed discussion of the different kinds of consistency can be found in Chapter 3 of the *Handbook of Constraint Programming* [16]. One of the most well-known types is called arc consistency (AC). Assuming the existence of a binary constraint between a pair of variables x and y, the domain of x is said to be arc consistent if, for every value a in the domain of x, there exists at least one value, b, in the domain of y such that the assignment $x = a, y = b$ is allowed by the constraint [54]. For example, suppose a particular CSP has two variables, x and y, each with domain $\{2, 3\}$, and a constraint stating that $x + y \leq 5$. We see that the domain of x is arc consistent because if we assign either 2 or 3 to x, we can find a value for y which will allow the relation $x + y \leq 5$ to be satisfied. Similarly, the domain of y is arc consistent with respect to the given constraint. Now, suppose the domains of x and y are actually $\{2, 3, 4\}$. An arc consistency algorithm will recognize that if we assign the value 4 to x, there is no value for y which would ensure that $x + y \leq 5$. Therefore, the algorithm would remove the value 4 from the domain of x. Similarly, there can be no solution to the given constraint if $y = 4$, and the domain of y would be reduced to $\{2, 3\}$. At this point, the given constraint is arc consistent [16]. Most constraint solvers enforce arc consistency on some but not all binary constraints [82].

The equivalent of arc consistency for non-binary constraints is usually referred to as generalized arc consistency (GAC). Informally, a domain D_i of variable x_i is said to be GAC (with respect to a specific constraint C_j) if, for every value in D_i, there exists an assignment of values to the remaining variables in $scope(C_j)$ which allows the constraint to be satisfied. A more formal definition of GAC is presented in the work of Bessiere [16]. As an example, suppose a CSP consists of three variables, x, y and z, where the domain of both x and y is $\{2, 3\}$, and the domain of z is $\{1, 2\}$. Suppose there is a constraint among the three variables: $x + y + z \leq 5$. It can be seen that assigning 1 to z can lead to solution $x = 2, y = 2, z = 1$ but assigning 2 to z cannot lead to any assignment of both x and y that would satisfy the constraint. The value 2 is not consistent with the given constraint and can be removed from the domain of z. The reduced domain of z, $\{1\}$, is generalized arc consistent. In order to enforce GAC on domains of all variables involved in constraint $x+y+z \leq 5$, all domains have to be examined. Looking at the domain of x, we see that assigning $x = 2$ can lead to solution $x = 2, y = 2, z = 1$ and assigning $x = 3$ cannot lead to any solution. The value 3 can therefore be removed from the domain of x. The domain of x is now $\{2\}$, the domain of y is $\{2, 3\}$ and the domain of z is $\{1\}$. Examining the domain of y, we see that the value 2 can lead to a solution but 3 cannot. Pruning the domain of y, we are left with singleton domains which are consistent with each other and constitute the only solution to the given constraint in this trivial CSP.

Although in this trivial example achieving GAC on all domains led to a unique solution, this is not generally the case. In fact, enforcing GAC may lead to singleton domains (proving the existence of a unique solution to the CSP), an empty domain for some variable (proving that no solution for the CSP exists), or domains which are all non-empty and some of which have more than one value left (in this case, the CSP may have many, one or no solutions). Additionally, the example illustrates that the enforcement of GAC may require checking a large number of combinations of values and may take a long time in a complex problem with many constraints. Consequently, GAC is not usually enforced on non-binary constraints, except for a special class of constraints, called global constraints, which have specialized effective propagation

algorithms [82].

Global constraints are constraints representing a complex relationship among a non-fixed number of variables [98]. One of the most well-known global constraints is the *AllDifferent* constraint, which states that the values assigned to a set of variables have to be distinct. It can alternatively be represented by a set of binary inequality constraints. The *AllDifferent* constraint possesses a very effective propagation algorithm based on the problem of finding the maximal matching in a bipartite graph [69, 16, 98]. This algorithm allows a high level of consistency to be achieved. Similarly, other global constraints have algorithms associated with them that allow a greater amount of propagation to be achieved than if a large number of simple constraints were used to represent the same restrictions.

Although it may be possible to solve some problems using only propagation algorithms, such an approach is likely to be very time-consuming. Therefore, propagation is usually used as a pre-processing step to achieve some target level of consistency and then throughout search when an assignment of a value to a variable implies that domains have to be reduced further in order to maintain this level.

2.1.1.2 Systematic Search

Search is performed by assigning a value to a particular variable, branching when several values are possible for this variable. The order in which variables are assigned (variable ordering heuristic) and the order in which values are tried for each variable (value ordering heuristic) play an important role in the efficiency of the search procedure [54]. Every time a value is assigned to a variable, propagation can be performed in order to eliminate values in the domains of other variables which are inconsistent with this assignment. If, at any time, all values in a variable's domain have been tried, or constraint propagation results in an empty domain for some variable, the search backtracks, un-assigning the value given to the variable at the previous node in the search tree. The search then continues by trying a different value from this variable's domain (according to the value ordering heuristic) or by trying to assign a different

variable (according to the variable ordering heuristic). When a leaf node is reached, and all variables are assigned values, search stops because a solution has been found [54, 82].

This is the general outline of how search is performed, and there are in fact very sophisticated methods for backtracking and also for determining the amount of propagation that has to be performed each time an assignment is made. An overview of such techniques is presented in the work of van Beek [95] and Prosser [67].

In order to solve a CSOP, a branch-and-bound approach is used. In particular, assuming that the CSOP is a minimization problem, the objective function $f(x)$ is incorporated into the model by placing a constraint of the form $f(x) < f(s)$ every time a solution s satisfying the constraints is found (in a maximization problem, the constraint $f(x) > f(s)$ is added). This implies that all further solutions must have a better objective function value. Branch-and-bound can prune sub-trees in the search tree based not only on the feasibility of a current partial assignment but also on its objective function value. Whenever it is not possible to find a solution satisfying all of the original problem constraints and the constraint on the objective function value, the last feasible solution found has been proven to be optimal [8]. This basic approach, together with some more sophisticated techniques, which are discussed below, will be used in the following chapters.

2.1.1.3 Additional Inference Methods

In addition to a variety of propagation and search techniques, there are other inference methods which are used in CP to reduce the size of the search space. Two such techniques are based on the ideas of dominance rules and shaving.

Dominance Rules A dominance rule, in an optimization problem, is a constraint that forbids assignments of values to variables which are known to be sub-optimal. That is, it can be shown that for any such assignment, a different solution exists which is of better or equal quality [10, 81]. Excluding sub-optimal assignments reduces the amount of search necessary to find

the optimal solution. For example, dominance rules are used by Smith [80] for the problem of designing an optical fibre network called SONET. The network consists of some number of client nodes and some number of rings which join these nodes. There are known demands between pairs of nodes. Each node is installed on a ring using an add-drop multiplexer (ADM), and the objective is to minimize the total number of ADMs required. In this problem, installing only one node on a ring is sub-optimal since the only reason to install a node is to have it comminicate with another one. However, solutions in which one node is placed on a ring are in no way forbidden by the constraints of the problem. A dominance rule is therefore added in order to ensure that each ring has at least two nodes placed on it. This results in the problem being solved more efficiently [80, 82].

Shaving Shaving is a consistency-enforcing procedure for CSPs. It is based on temporarily adding constraints to the problem, performing propagation and making inferences based on the resulting state of the problem [27, 96]. For example, a simple shaving procedure may be based on the assignment of a value a to some variable x. If propagation following the assignment results in an empty domain for some variable, the assignment is inconsistent, and the value a can be removed from the domain of x [27, 96]. In a more general case, both the temporary constraint and the inferences made based on it can be more complex. Shaving has been particularly useful in the job-shop scheduling domain, where it is used to reduce the domains of start and end times of operations [21, 57]. For such problems, shaving is used either as a domain reduction technique before search or is incorporated into branch-and-bound search so that variable domains are shaved after each decision [21].

2.1.2 Modelling

One of the advantages of constraint programming is its expressiveness as a modelling language, which is a result of the lack of restrictions on the types of variables and constraints that can be included in the model of a problem.

In particular, the domains of the variables of a CSP can be integer, binary, continuous, or can consist of sets, strings or other pre-defined objects [8]. Constraints may be non-linear and state complex relationships among variables. Moreover, constraints are declarative, meaning that they specify a relationship among variables without stating the algorithm that will be used to find values for these variables [8]. One example of such a constraint, which will be used in further chapters, states an implication relationship between two variables x_i and x_j: $x_i = a \rightarrow x_j = b$ (where a and b are in the domains of x_i and x_j, respectively) [81]. Another example is a meta-constraint, which combines several constraints, treating each one as a variable taking the value 1 if it is satisfied, and the value 0 if it is not satisfied. For example, the meta-constraint $(x_i = k) \leq (x_j = l)$ states that x_i can equal k only if x_j equals l.

The flexibility of CP as a modelling language implies that there exist many different ways to model a particular problem as a CSP or a CSOP. In fact, as the literature shows, the way in which a problem is modelled may have a big influence on how efficiently it can be solved [82, 80, 79, 48, 7]. This is the case with the problems we discuss in Chapters 3 and 4.

In [82] and [81], Barbara Smith provides an overview of the major steps that need to be taken in order to model a problem as a CSP. The following list is a brief summary of these major steps.

1. **Choose a viewpoint.** A viewpoint is determined by the set of variables and their corresponding domains that are chosen to represent a problem; these have to be meaningful in terms of the actual problem being modelled, and complete assignments of all variables should include all possible solutions of the problem. Different viewpoints lead to different models, and how a problem is modelled has a direct impact on how quickly it will be solved.

   Naturally, there are usually several viewpoints for a particular problem. The viewpoint chosen for representing a problem should be the one that allows the use of constraints which have effective propagation algorithms (e.g. global constraints) and which allow for a concise statement of the problem. Combinations of viewpoints may also be used.

2. **Write the Constraints.** Clearly, the constraints should be expressed in a way which will ensure that solutions of the CSP are in fact solutions of the problem. However, additional considerations have to be taken into account when constraints are added to the model. In particular, since the manner in which a constraint is expressed determines how propagation is going to be performed, it is important to choose a constraint representation for which an effective propagation algorithm exists. For example, the *AllDifferent* constraint could be alternatively formulated using a set of binary $\neq$ constraints. However, it is known that the *AllDifferent* constraint has an efficient propagation algorithm based on the idea of maximum matchings in graphs [97] and is likely the better choice.

   In order to express constraints more concisely, constraints with the same scope may be combined and variables may be eliminated by substitution using other constraints. Global constraints, extensional constraints (which allow the user to explicitly state the tuples allowed by the constraint) and meta-constraints could also be used for this purpose.

3. **Consider the Possibility of Adding Auxiliary Variables.** Such variables are introduced into the model either when it is difficult to express all of the constraints in terms of existing variables, or in order to allow the constraints to be expressed in a form that would allow more propagation. In both of these cases, adding auxiliary variables will lead to a more effective model. In order to determine whether auxiliary variables may be helpful, and how they should be defined, one usually needs to carefully examine the constraints of the model and also consider alternative viewpoints. One method that allows auxiliary variables to be created is hidden variable transformation, which is used to convert a non-binary CSP into a binary CSP. This latter technique is discussed by Bacchus & van Beek [4].

   It may seem initially unintuitive that adding more variables could result in a better model. However, one should always keep in mind that constraint programming reasons about constraints and variable domains, and that relevant additional variables may lead to more

domain reductions, and in turn, more effective methods for solving the problem.

The use of auxiliary variables is demonstrated by Smith et al. [83], for example, who address the Golomb ruler problem. This problem is known to be hard and is part of the CSPLib benchmark library (http://www.csplib.org) [83]. A Golomb ruler is a ruler with m marks which are set in such a way that the differences between any two of the m marks are distinct. The goal of the problem is to find a ruler of smallest length with a specified number of marks. The problem can be modelled by a set of variables x_i for $0 < i \leq m$. Since the constraints of the problem restrict the differences between two variables x_j and x_i for $0 < i < j \leq m$, a set of auxiliary variables d_{ij} representing the differences $x_j - x_i$ can be added. The experiments of Smith et al. [83] show that this addition results in more effective models.

4. **Consider the Possibility of Adding Implied (Redundant) Constraints.** Constraints which are implied by the existing constraints of the model and which, therefore, do not change the solutions resulting from this model, can be added in order to increase propagation and reduce the search effort. A necessary, but not sufficient, condition for an implied constraint to be useful is that it forbids some assignment to the variables that is not forbidden by the constraints of the model but that cannot be part of any solution.

   Implied constraints are used, for example, in the optical fibre network design problem discussed by Smith [80] (also mentioned above in Section 2.1.1.3). In this problem, nodes have to be placed on rings of the network, and rings have some specified capacity. Each node is "connected" to some other nodes. If some node has to be connected to a greater number of nodes than the capacity of any single ring, then it has to be placed simultaneously on two rings. This is not taken into account by the original problem constraints but has to be true in any solution. Therefore, implied constraints stating this restriction can be added to the model.

   It can also happen that an implied constraint is useful simply because a suitable global

constraint has not been found. In fact, if a large number of similar problems possesses a structure that may benefit from the addition of redundant constraints, it is better to implement a new global constraint. If the problem does not belong to a large class of problems, then one should look for good redundant constraints, which should usually involve only a small number of variables (be of small arity).

Work has been done by Hnich et al. [47] and Frisch et al. [38] on trying to create "automatic" ways of generating implied constraints. However, implied constraints are often developed (because search is taking an extremely long time) by examining the search tree in detail, finding assignments that are obviously wrong and then writing constraints that would eliminate these assignments from consideration.

5. **Consider Reformulating the CSP.** After some models have been developed, one could try looking at the problem from a different viewpoint either by using a standard transformation or by viewing the problem from a different angle. Standard transformations include non-binary to binary translations with the use of hidden variables or the creation of a dual in which original constraints are replaced by new variables and binary constraints relating these dual variables (for constraints which have a non-empty intersection). Such transformations are discussed by Bacchus et al. [4] and Smith et al. [78].

   In some problems, such as the permutation problems studied by Hnich et al. [48], it is possible to switch the roles of variables and values. The resulting viewpoint is called the dual viewpoint, and the new variables are referred to as dual variables [7, 82]. Note that the term *dual* has two different meanings in constraint programming: in standard transformations of non-binary to binary problems, it is used to describe models in which the roles of constraints and variables are switched; in other contexts, it usually refers to a switch of the roles of variables and values. Throughout the rest of the dissertation, we use the term *dual* to mean the switch between the roles of variables and values. Such dual variables have been used to create more efficient models for the problems addressed

in the work of Smith [80, 79] and Hnich et al. [48].

6. **Try Combining Viewpoints.** One can create a model by combining two or more mutually redundant viewpoints. This can yield two benefits. Firstly, some (but not all) constraints may be easier to express in terms of one set of variables than the other, possibly leading to a simpler model in spite of the increased number of variables. Secondly, some constraints may propagate better when expressed in terms of one set of variables, and others when expressed in terms of another one. Therefore, combining several viewpoints may allow each constraint to be expressed in a way that would lead to the greatest amount of propagation.

   In order to ensure that assigning values to the variables of one viewpoint will result in assignments of variables in other viewpoints, channelling constraints have to be included. These constraints state relationships between the variables of different viewpoints. Thus, although the total number of variables is increased, only half of these variables have to actually be assigned during search–the rest will be assigned via propagation of the channelling constraints.

   As an example, consider the Golomb ruler problem (mentioned above in step 3). The goal of the problem is to find a smallest-length ruler with m marks, such that the differences between any two marks are distinct. The problem can be modelled by a set of variables x_i, for $1 \leq i \leq m$, each representing the position of mark i on the ruler. The necessary constraints are $|x_j - x_i| \neq |x_l - x_k|$ for all pairs of variables x_i, x_j and x_l, x_k, and $x_i < x_{i+1}$ for all $i < m$, and $x_1 = 0$ [83]. Alternatively, the problem can be modelled using variables d_{ij}, which represent the distance between two marks on the ruler, rather than the position of the marks. The constraints that need to be included in order to model the problem using this second viewpoint are $d_{ik} = d_{ij} + d_{jk}$ for $1 \leq i < j < k \leq m$, and an *AllDifferent* constraint on the set of d_{ij} variables, $\forall i, j \in \{0, 1, \ldots, m\}$. The two viewpoints can be combined using the channelling constraints $x_j - x_i = d_{ij}$ for

$1 \leq i < j \leq m$ [82]. As a result of these constraints, when values are assigned to the x_i variables, the d_{ij} variables also get assigned, and vice versa. Note that the model discussed in step 3 also uses both sets of variables, but constraints are expressed only in terms of d_{ij}s, which are considered to be auxiliary variables [82].

The choice of variables that will be used during search is an important issue to consider when several viewpoints are employed. One may choose to search using only one set of variables, or several. The latter approach may be particularly effective when combined with a dynamic variable ordering heuristic. When search is performed, such a heuristic chooses the next variable to assign based on the current state of the domains. The heuristic has more variables to choose from when viewpoints are combined, and therefore, has more opportunities for making an effective choice [82].

7. **Reduce Symmetry.** Symmetry is introduced into a model when distinct variables and/or values are used to represent indistinguishable entities of the problem being modelled [82]. This implies that several different solutions of the CSP or CSOP may correspond to the same solution to the problem and unnecessarily increases the size of the search space. Symmetry can often be avoided by choosing a different viewpoint or by including symmetry-breaking constraints [82, 77].

   In the Golomb ruler problem [83], for example, reflection symmetry is present: reversing the positions of the marks yields an essentially equivalent solution to the problem. Consider an optimal Golomb ruler of length 6. This ruler has marks at positions 0, 1, 4 and 6, or at positions 0, 2, 5, 6 [24]. The two rulers are equivalent since the difference between marks 1 and 2 in the first is the same as the difference between marks 3 and 4 in the second, the difference between marks 2 and 4 in the first is the same as the difference between marks 1 and 3 in the second, and the differences between marks 1 and 4, and 2 and 3, are the same in the two rulers. It can be seen that if one of these rulers is found, then the other can be easily generated. Consequently, in solving the problem, the

symmetry-breaking constraint $|x_2 - x_1| < |x_m - x_{m-1}|$, or $d_{12} < d_{m-1,m}$, can be used to ensure that only one type of solution is found, reducing the size of the search space [83].

This review of the major steps involved in modelling problems as CSPs and CSOPs demonstrates that in order to create an effective CP model, a deep understanding of the underlying problem is necessary. Additionally, most questions discussed above, such as the choice of viewpoint or choice of search variables, can only be answered after much experimentation with the available options. However, the effort put into finding a good model will likely pay off in the form of an effective approach to solving a hard problem.

2.1.3 Problems Involving Uncertainty and Change

In traditional constraint satisfaction, it is assumed that all variables, domains and constraints are well-defined prior to an attempt at solving the problem. However, in reality, problems may arise in which the definition of the problem is somehow uncertain or may be subject to change over time. Such problems require a different set of approaches and algorithms than standard CSPs. An overview of these approaches is presented in Chapter 21 of the *Handbook of Constraint Programming* [19] and in the survey paper of Verfaillie & Jussien [99].

Incorporating uncertainty and change has led to many new types of CSPs, such as *fuzzy*, *mixed*, *uncertain*, *probabilistic*, *dynamic*, *recurrent*, *branching* and *stochastic* CSPs. According to Brown & Miguel [19], the first four of these may be classified as "Uncertain Problems", while the rest may be described as "Problems that Change". We adopt this classification in presenting brief descriptions of these different types of CSPs below.

2.1.3.1 Uncertain Problems

In some cases, there may be inherent uncertainty present in the definition of a problem. Firstly, the problem may be imprecise, in the sense that it may allow constraints to be partially satisfied, totally satisfied or totally unsatisfied [19]. Such problems are usually referred to as fuzzy CSPs.

Secondly, the problem may not be completely defined at the time when it has to be solved. In particular, one may not know the complete set of variables and/or constraints, and variable domains may not be completely specified [19]. Such problems are usually modelled as mixed CSPs [32] or probabilistic CSPs [30].

Fuzzy CSPs A constraint in the classical definition of a CSP can be viewed as a set of possible combinations of values that are allowed to be assigned to the variables in the scope of this constraint. In fuzzy CSPs, a constraint can be represented as a fuzzy set with a membership function which associates a degree of satisfaction to each possible assignment of values to variables from the corresponding domains. The degree of satisfaction is usually a value between 0 and 1, with 0 representing complete violation of the constraints, and 1 representing complete satisfaction of the constraints [19, 58]. The goal of the problem is to find an assignment with the maximum overall degree of satisfaction [19], which is usually defined to be the minimum over the satisfaction degrees of all constraints [28]. In addition to being able to deal with uncertain parameters in constraints, the fuzzy CSP framework allows one to represent the idea that one solution may be preferable to another, and some constraints may be more important than others [28].

Mixed CSPs In a Mixed CSP, the variables of the problem can be divided into two sets: ones that can be assigned by the solver, and ones which are controlled by some external source, such as a user or a random process [19, 32]. As stated by Fargier et al. [32], the definition of a solution to a mixed CSP is based on the assumptions made about the initial problem formulation. It is assumed that there is a deadline by which a solution to the problem has to be found. It may happen that all of the initially-unknown information about the problem will be revealed prior to, possibly just before, the deadline. Consequently, it is a good idea to compute a *conditional* solution, one that states which values should be assigned to the decision variables given a particular realization of the uncontrollable parameters. On the other hand, it may be possible that no new information will be revealed prior to the deadline. In this case, a

pure solution is required. A pure solution should be a solution to one or more of the possible realizations of the problem [19]. There is also the possibility of observing the values of some of the uncontrollable parameters prior to the deadline, leading to a problem definition between the two extremes.

Uncertain CSPs Uncertain CSPs have been created in order to handle problems in which the coefficients in some or all of the constraints are uncertain. The goal in such problems is to either find a set of solutions which would contain at least one solution for each possible realization of the uncertain parameters, or the most robust solution, which will be consistent in the greatest number of realizations [19, 103].

Probabilistic CSPs In the first type of probabilistic CSPs, each constraint has some probability of being active in a particular realization of the problem [19, 30]. Such probabilistic CSPs are closely related to the research area of soft constraints, which model desired properties, or preferences, rather than being strict restrictions that cannot be violated [58]. This is because in both cases, some of the constraints are allowed to be violated under some conditions. In the second type of probabilistic CSPs, the variables are divided into uncontrollable and controllable, as in mixed CSPs. However, unlike in mixed CSPs, there is a probability distribution associated with the uncontrollable parameters [31]. In both types of probabilistic CSPs, the goal is to find an assignment of values to variables having the highest probability of being a solution to the actual realization of the problem.

2.1.3.2 Changing Problems

In many real-world applications, the definition of the problem may be revealed with time or may change, and there may exist opportunities for reacting to the availability of new information or the changes that have occurred [19]. Such problems can be represented in a variety of ways in constraint programming.

Dynamic CSPs Dynamic CSPs are sequences of standard CSPs in which each problem in the sequence can be obtained from the previous one by an addition or removal of a constraint [26, 19]. The goal of such problems may be to find robust solutions, which have the highest probability of remaining solutions after the occurrence of a change, to minimize the cost of a change if it is required, or to minimize the reaction time by finding a new solution as quickly as possible [19]. More details on such CSPs can be found by consulting the literature survey of Verfaillie & Jussien [99].

Recurrent CSPs Recurrent CSPs are used to model problems in which changes are either temporary or recurring [19, 100]. Although it is assumed that no knowledge about the changes is known in advance, this knowledge may be learned in the process of solving the problem. The reader is referred to the papers of Wallace & Freuder [100], Minton et al. [59] and Hebrard et al. [43] for further information.

Branching CSPs In branching CSPs, it is assumed that the problem evolves with time, and, in particular, new variables and constraints associated with these variables are added to the problem periodically [34, 19]. Which variables and constraints will be added to the problem is unknown at the time when the problem is solved. However, it is assumed that some probabilistic model of future additions is available. Each variable has a utility associated with it, and can be assigned a value when it arrives (be accepted), or can be left unassigned (be rejected). In the first case, the utility of the variable is counted in the calculation of the overall utility of the assignment. In the second case, the utility of the variable is not counted. A solution to a branching CSP is a policy which states, for every possible sequence of additions to the problem that may occur, what actions should be taken (which variables should be accepted or rejected). In particular, the goal of branching CSPs is usually to find a policy which attains the maximum overall utility and satisfies all of the constraints [34]. Branching CSPs are very closely related to Markov Decision Processes [33], since both model a process which is defined by a set of states and a set of probabilities of moving between states which depend only on the current

state and the action being taken [35].

Stochastic CSPs In stochastic CSPs, the variables are divided into two sets: a set of decision variables, which can be assigned values, and a set of stochastic variables, which can only be observed. Stochastic CSPs also usually include probabilistic constraints which are required to be satisfied by only a fraction of solutions [90, 5]. Each constraint which involves some stochastic variables is called a chance constraint and has an associated parameter which specifies the percentage of scenarios in which it has to be satisfied [101, 90]. A scenario is defined by the values assigned to the stochastic variables. Walsh [101] and Balafoutis & Stergiou [5] take a policy-based approach to such problems and extend the traditional ideas of systematic search to deal with stochastic variables. Manandhar et al. [55] and Tarim et al. [90] propose an alternative approach based on the idea of decomposing the problem into many scenarios. Other papers on stochastic CSPs include those of Benoist [12], Bordeaux & Samulowitz [18] and Tarim & Miguel [91].

2.2 Optimal Design and Control of Queues

Although constraint programming is able to deal with a variety of problems involving uncertainty, as mentioned above, it has yet to build closer links with such research areas as queueing theory [19]. The main goal of this dissertation is to take a step towards creating this link by demonstrating that constraint programming can be applicable to queueing control and design problems. In this section, we discuss some of the major work in this wide area of research based on Chapter 6 of the book *Fundamentals of Queueing Theory* [42] and the overview paper of Tadj & Choudhury [87]. A detailed summary and bibliography of the earlier work on the design and control of queues is presented in a paper by Crabill et al. [25].

It should firstly be noted that much of queueuing theory has dealt with the development of descriptive models, that is, ones which are able to provide characteristics of a particular queue, such as the expected length of the queue, the expected waiting time of customers in the

queue, the proportion of lost customers if an upper bound on the length of the queue is placed and others [42, 87]. However, queueing theory also gives rise to many natural optimization questions regarding the design and control of queues.

Both queueing design and control problems deal with finding the optimal values for some of the parameters of the queue, usually referred to as the *controllable* parameters [42]. These parameters may be: the number of servers (channels) available for serving arriving customers, the limit on the length of the queue(s) (system capacity), the arrival rate of customers to the queue(s), the service rates of the server(s), as well as any combination of these [42]. Queueing design problems are static–once the optimal value of a controllable parameter is determined, it becomes a fixed characteristic of the queue. Queueing control problems, on the other hand, are dynamic–the goal in such problems is usually to determine an optimal action to take when the queue is in a particular state. The state can be defined in a variety of ways based on the current characteristics of the queue such as the queue length or the customer arrival rate. It should be noted that although most problems studied either deal with control or design of a queueing system, these two questions may, in reality, be inter-related and may not be solved separately [25]. Examples of problems involving both the optimal design and the optimal control of a queue are discussed in Section 2.4 and Chapter 4.

In addition to being subdivided by the choice of the controllable parameter, queue design and control problems can be categorized according to the assumptions made about the *uncontrollable* parameters. Thus, problems may arise which deal with specific arrival or service time distributions, specific number of servers, or specific order in which customers are served. The choice of controllable parameter(s) and the particular assumptions made about the uncontrollable ones result in many possible variations of queueing design and control problems. For example, one may consider the queueing design problem of finding the optimal number of servers to employ in a service facility under the assumption of Poisson arrivals, exponential service times and first-come, first-serve queue discipline. One may then look at the related queueing control problem of determining when to increase and decrease the number of workers

available to serve customers (e.g. workers may be switched from another type of job) depending on the length of the queue, under the same assumptions. If we assume that the order in which customers are served does not follow the simple first-come, first-serve discipline, while the rest of the problem assumptions stay the same, two new problems arise. Similarly, if we change our assumptions of the arrival or service time distribution, we will get a new queue design problem and a new queue control problem. Queueing design and control problems may therefore be classified according to many different criteria: the arrival distribution assumptions, the service time distribution assumptions, the number of servers, the costs associated with the operations of the queue, etc. We refer the reader to the overview papers of Tadj & Choudhury [87] and Crabill et al. [25] for more complete lists of variations of design and control problems involving queues.

In the following two sub-sections, we discuss the major works in queue design and queue control with different controllable variables, with the implication that the same optimization problems may arise when different assumptions are made regarding the queue's (uncontrollable) characteristics.

2.2.1 Optimal Design of Queues

One of the most fundamental papers in queueing design is that of Hillier [46]. It discusses queueing design problems in the context of an industrial setting where a balance between the cost of service and the cost of waiting for that service has to be found. Three related problems are examined. It is assumed that there is an infinite source of customers, that all arrivals will join the queue and remain in it until they get served, that there is no limit on the queue length and that the queue is in a steady state. The models are general enough to be applicable regardless of the assumptions on the service and arrival distributions and on the queue discipline.

In the first problem studied by Hillier [46], it is assumed that both the service rate and the arrival rate are fixed, and one needs to determine the optimal number of servers required to minimize a linear cost function consisting of the total cost of service and the total cost of

customer waiting time.

In the second problem, the decision variables are both the number of servers and the arrival rate. This problem may arise in the context of locating service facilities throughout a population. For example, one may need to determine the number of medical centers that need to be placed throughout a particular area, the number of doctors at each medical center and the proportion of the population that should be assigned to each particular center (service facility). Determining the latter quantity is essentially equivalent to finding the arrival rate. The objective is to minimize the expected total cost per unit serviced, and it is assumed that the arrival rates and number of servers at each facility are equal. Firstly, Hillier proves that one service facility is optimal for the whole population. Secondly, it is observed that this result makes sense only if customer travel time to the facility is not considered in the model. Several models which incorporate travel time are further presented.

In the third problem, the goal is to determine both the number of servers and the service rate that would minimize the overall cost of service and customer waiting time. Stidham [84] and Morse [61] also concentrate on problems in which one needs to determine the optimal service rate and number of servers. These authors prove the optimality of the solution in which there is only one server under different arrival and service time distributions.

Other papers deal with systems which serve groups of customers rather than single customers [73], situations when the server may not be able to serve customers for periods of time [51] and systems with impatient customers, who leave without getting served if they find the server(s) busy [56]. For a detailed review of the work done on optimal design of queues, the reader is referred to the paper of Tadj & Choudhury [87].

2.2.2 Optimal Control of Queues

The term *policy* is used in queueing control literature to describe a rule which prescribes, given a particular queue state, the actions that should be taken in order to control the situation in the queue. Most of the research on the optimal control of queues has focused on determining

when a particular type of policy is optimal, rather than on finding the actual optimal values of the controllable parameters which define the policy [42]. Note that the term *optimal policy* is used in the literature to mean both the optimal *type of policy* and the optimal *parameter values* for a given policy type. The distinction between the two is important since showing that a particular policy type is optimal is a theoretical question, whereas finding the optimal values for a specific policy type is a computational one. The focus of this dissertation is on the latter question. Therefore, the term *optimal policy* is used throughout the dissertation to refer to the optimal numerical parameters for a pre-defined policy type.

The earliest papers in the area of queue control are those of Romani [70] and Moder & Phillips [60], which both deal with controlling the number of servers based on the queue length. Romani [70] considers a situation where the queue is never allowed to grow beyond some number M. In particular, the number of channels (servers) is increased by one every time queue length becomes greater than M, and there is no upper bound on the possible number of open channels. The channels are removed ("cancelled") when the queue becomes empty. The work of Moder and Phillips [60] considers a policy that is defined by two parameters, v and N, with the interpretation that each time the number of customers in the queue reaches N, the number of channels is increased by 1, and when the number of customers in the queue drops to v and a service completion occurs, the number of channels is decreased by 1. There are upper and lower bounds on the number of channels that can be operated at any time. It is assumed that workers who are not serving customers at a particular time point are performing some other tasks which are not directly dependent on the arrival process of customers. This paper presents several derivations of steady-state probabilities of having a particular number of customers in the queue given a (v, N) policy as well as derivations of the quantities of interest such as the expected waiting time.

The question of finding the optimal number of servers for each system state is also addressed by Hersh [44]. It is assumed that workers have to serve customers (perform "queue" work) and also complete other tasks ("non-queue" work), which are not dependent on customer

arrivals. Costs are incurred due to the waiting time of customers, the uncompleted non-queue work that has to be completed in overtime and the switching of workers between the two types of tasks. A policy is given by parameters (v, N) so that when there are v customers in the system and one of them gets served, the worker who just completed service gets switched to performing fixed non-queue work, and when the number of customers in some queue reaches N, a worker gets switched to queue work, opening a new service channel.

In the paper of Yadin and Naor [102], the controllable parameter is the service rate of a single server. Given a feasible set of service capacities $\{\mu_0, \mu_1, \ldots, \mu_k, \ldots\}$, with $\mu_0 = 0$ and $\mu_{k+1} > \mu_k$, the class of policies considered is defined by two integer vectors $\{R_k\} = \{R_1, R_2, \ldots, R_k, \ldots\}$ and $\{S_k\} = \{S_0, S_1, \ldots, S_k, \ldots\}$ with $R_{k+1} > R_k$, $S_{k+1} > S_k$, $R_{k+1} > S_k$ and $S_0 = 0$. The interpretation given to these two vectors is that whenever the size of the system reaches R_k, the service rate is increased from μ_{k-1} to μ_k, and whenever the size of the system is reduced to S_k, the service rate μ_{k+1} is switched to service rate μ_k. The authors derive some quantities characterizing the queue if this policy is used, such as the expected system size.

Problems involving control of arrivals into the system are investigated by Stidham [85] and Rosenshine & Rue [71]. Serfozo & Lu [74] consider a problem involving a single-server system, with both the arrival rate and the service rate being the decision variables. A finite set of arrival and service rate pairs is given, and a policy is determined by observing the queue length at a particular decision epoch and choosing the appropriate pair. The objective function is based on switching costs, which are incurred every time a service rate/arrival rate pair is chosen at a decision epoch which is different form the previous service/arrival rate pair, and usage costs, which are based on chosen service rates and queue length. Two other papers dealing with essentially the same type of policy are those of Gebhard [40] and Heyman [45].

2.2.3 Methods for Solving Queueing Design and Control Problems

It should be noted that much of the literature on queueing design and control problems focuses on modelling approaches and theoretical questions. Very few papers discuss the development of methods for finding the actual optimal values for the controllable parameters. We mention some of these papers in this section.

In general, queueing design (numerical) optimization problems can be solved using differential calculus and linear, integer or non-linear programming [42], as well as heuristics. One example of work on numerical, rather than strictly theoretical, questions is the paper by Grassmann et al. [41], in which the problem of optimizing the service rate in a one-server queueing system with exponential inter-arrival times is addressed using differentiation of various quantities.

Queueing control optimization problems have usually been solved via dynamic programming and techniques used for Markov decision problems such as value or policy iteration [42]. For example, in the work of Nobel & Tijms [62], a problem with two heterogeneous servers is considered, with one server being faster than the other. The faster server is always 'on' whereas the slower server can be turned 'on' and 'off' depending on the queue length. A specialized policy iteration algorithm is presented for determining when the slower server should be turned on and off.

More recently, some papers have appeared which use linear programming to solve queueing control problems. Examples of such work are the papers by Fan-Orzechowski & Feinberg [29] and Berman & Sapna-Isotupa [14].

Berman & Sapna-Isotupa [14] consider a retail facility with a back room and a front room. Work in the front room is generated by the stochastic process of customer arrivals who form a queue if there are not enough workers to serve them. The amount of back room work is correlated with the amount of work in the front room, with each front room customer generating one back room job. The problem is formulated as a linear program, and some experimental results are presented, highlighting the efficiency of the proposed method. This problem is very

similar to the problems of interest in this dissertation (see Section 2.4). The major difference is that, in this dissertation, the amount of work in the back room is assumed to be known, whereas in the work of Berman & Sapna-Isotupa [14] back room work is assumed to be a direct consequence of front room work. In Section 6.2.1.2, we mention a similar problem that would be interesting to investigate in the future.

Fan-Orzechowski & Feinberg [29] study a queueing system with m customer types, exponential service and inter-arrival times, c servers and a capacity of N. The goal of the problem is to decide, every time a customer arrives, whether to accept or reject this customer. A cost is incurred every time a customer is rejected, and a reward is gained every time a customer is accepted. The objective is to maximize the average reward per unit time subject to a constraint requiring the average penalty cost for rejected customers to be bounded from above. A linear programming approach is employed. Note the similarity of this problem with Branching CSPs discussed in Section 2.1.3.2. Exploring the similarities of these problems as well as of approaches for solving them would be an interesting direction for further work.

Additionally, some queueing control problems are solved using heuristics. An example of such a problem appears in the work of Berman & Larson [13], which is discussed in Section 6.2.2.2 as it is an extension of the problem of interest in Chapter 3.

2.3 Cross-training Literature

The two queueing optimization problems addressed in this dissertation deal with a retail facility which employs cross-trained workers. Questions related to cross-training, such as the optimal number of cross-trained workers to hire, or the number of skills that workers should be cross-trained in, have been addressed by many researchers.

Brusco & Johns [20] use integer linear programming in order to help evaluate different cross-training options at a paper-mill. Each worker is assumed to be able to perform several types of jobs, and the number and types of jobs determine his/her skill class. The solution of

the integer program determines the number of employees from a particular skill class j that should be working on task k in planning period i.

Integer programming is also used by Slomp et al. [76], where the objective is to balance the workloads of cross-trained workers in a manufacturing cell. In particular, an integer program is used to find out how to assign workers to machines as well as to determine for which tasks/machines particular workers should be trained.

Cezik & L'Ecuyer [22] and Chevalier & Tabordon [23] examine cross-training issues of a call center. In the work of Cezik & L'Ecuyer [22], a complex method based on a combination of simulation with linear programming, cut generation and heuristics is used. It is assumed that call center agents are able to answer multiple types of calls, and the types of calls that the agent is able to handle determines his/her type. The goal of the problem is to determine the number of agents of each type that should be scheduled to work during a particular time period in order to minimize the operating costs of the call center while meeting some service level constraints. Chevalier & Tabordon [23] seek a staffing configuration which minimizes the proportion of customers who leave without getting served. An approximation for this quantity is proposed and evaluated via simulation. Simulation is further used to compare various staffing configurations.

Pinker et al. [65] deal with evaluating the trade-offs between specialized and cross-trained workers. It is stated that what is gained in efficiency when cross-trained workers are hired may be lost in quality. Thus, their approach combines a stochastic service center model with models of tenure-based and experience-based service quality. The paper demonstrates that the optimal staff mix depends both on the size of the system and the learning rates of the workers.

2.4 Two Related Problems of Berman et al.

The work of Berman et al. [15] concerns retail facilities, such as stores or banks, which have back room and front room operations. In the front room, workers have to serve arriving cus-

tomers, and customers form a queue and wait to be served when all workers are busy. In the back room, on the other hand, work is less time-sensitive and may include such tasks as sorting or processing paperwork. All workers in the facility are cross-trained and are assumed to be able to perform back room tasks equally well and serve customers with the same service rate. Therefore, it makes sense for the managers of the facility to switch workers between the front room and the back room depending both on the number of customers in the front room and the amount of work that has to be performed in the back room. Berman et al. study two related problems. In the first problem, the goal is to find a switching policy that minimizes the expected customer waiting time in the front room, subject to a constraint requiring the completion of all work in the back room. In the second problem, the objective is to find the smallest number of cross-trained workers required in the facility in order to satisfy a quality of service constraint in the front room and a constraint requiring all of the back room work to be performed.

It can be seen that both of these problems fall into the class of queueing design and control problems and attempt to answer similar questions to the ones of interest in the cross-training literature.

The first problem of Berman et al. and an extension of their second problem are the focus of this dissertation. In this section we therefore provide a detailed description of the work done by Berman et al. [15].

2.4.1 Berman et al.'s Problem P_1

Let N denote the number of workers in the facility, and let S be the maximum total number of customers allowed in the front room at any one time.[1] This implies that when there are S customers present, arriving customers will be blocked from entering the front room. Customers arrive according to a Poisson process with rate λ . Service times by workers in the front room follow an exponential distribution with rate μ. In order to complete all the back room work, there is a known minimum requirement, B_l , on the expected number of workers in the back

[1]The notation used by Berman et al. [15] is adopted throughout this dissertation.

room. Only one worker is allowed to be switched at a time, and both switching time and switching cost are assumed to be negligible.

The goal of problem P_1 is to find an optimal approach to switching workers between the front room and the back room so as to minimize the expected customer waiting time, denoted W_q , while at the same time ensuring that the expected number of workers in the back room is at least B_l. Thus, a policy needs to be constructed that would specify how many workers should be in the front room and back room at a particular time, and when switches should occur.

2.4.1.1 Model

Berman et al. define a policy in terms of quantities k_i, for $i = 0, \ldots, N$. This policy states that there should be i workers in the front room whenever there are between $k_{i-1} + 1$ and k_i customers in the front room, for $i = 1, 2, \ldots, N$. As an illustration, consider the policy $(k_0, k_1, k_2, k_3) = (0, 2, 3, 6)$, with $N = 3$. This policy states that when there are $k_0 + 1 = 1$ or $k_1 = 2$ customers in the front room, there is one worker in the front room; when there are 3 customers, there are 2 workers; and when there are 4, 5, or 6 customers, all 3 workers are employed in the front. Alternatively, k_i can be interpreted as an upper bound on the number of customers that will be served by i workers under the given policy. Yet another interpretation of this type of switching policy comes from noticing that as soon as the number of customers in the front room is increased by 1 from some particular switching point k_i, the number of workers in the front room changes to $i + 1$. This definition of a policy forms the basis of the model proposed by Berman et al., with the switching points k_i, $i = 0, \ldots, N - 1$, being the decision variables of the problem, and k_N being fixed to S, the capacity of the system. In Berman et al.'s work, an optimal policy is a set of values for the switching points, k_i, which minimizes the expected waiting time subject to the back room constraint. In other words, the paper of Berman et al. [15] is not concerned with proving the optimality of the policy type specified by the k_is, but rather with the determination of optimal numerical values for the quantities. In the following sections, the term *optimal policy* therefore refers to the optimal set of values for the

k_is.

In order to determine the expected customer waiting time in the front room and the expected number of workers in the back room given a policy defined by particular values of k_i, Berman et al. first define a set of probabilities, $P(j)$, for $j = k_0, k_0 + 1, \ldots, S$. Each $P(j)$ denotes the steady-state (long-run) probability of there being exactly j customers in the facility. Berman et al. define a set of balance equations for the determination of these probabilities:

$$\begin{aligned}
P(j)\lambda &= P(j+1)1\mu \qquad j = k_0, k_0+1, \ldots, k_1 - 1 && (2.1)\\
P(j)\lambda &= P(j+1)2\mu \qquad j = k_1, k_1+1, \ldots, k_2 - 1 && (2.2)\\
&\vdots \qquad\qquad \vdots \qquad\qquad\qquad \vdots \\
P(j)\lambda &= P(j+1)i\mu \qquad j = k_{i-1}, k_{i-1}+1, \ldots, k_i - 1 && (2.3)\\
&\vdots \qquad\qquad \vdots \qquad\qquad\qquad \vdots \\
P(j)\lambda &= P(j+1)N\mu \qquad j = k_{N-1}, k_{N-1}+1, \ldots, k_N - 1. && (2.4)
\end{aligned}$$

The probabilities $P(j)$ also have to satisfy the equation $\sum_{j=k_0}^{S} P(j) = 1$.

From these equations, Berman et al. derive[2] the following expressions for each $P(j)$:

$$P(j) = \beta_j P(k_0), \tag{2.5}$$

where

$$\begin{aligned}
\beta_j &= \begin{cases} 1 \quad \text{if} \quad j = k_0 \\ \left(\frac{\lambda}{\mu}\right)^{j-k_0} \left(\frac{1}{i}\right)^{j-k_{i-1}} X_i \quad \text{if} \quad k_{i-1}+1 \le j \le k_i \quad i = 1, \ldots, N \end{cases}, \\
X_i &= \prod_{g=1}^{i-1} \left(\frac{1}{g}\right)^{k_g - k_{g-1}} \qquad (2.6)\\
&\quad (X_1 \equiv 1), i = 1, \ldots, N.
\end{aligned}$$

[2]For details regarding these derivations, the reader is referred to Berman et al.'s paper [15].

$P(k_0)$ can be calculated using the following equation, which is derived by summing both sides of Equation (2.5) over all values of j:

$$P(k_0)\sum_{j=0}^{S}\beta_j = 1. \tag{2.7}$$

All quantities of interest can be expressed in terms of the probabilities $P(j)$. Expected number of workers in the front room is

$$F = \sum_{i=1}^{N}\sum_{j=k_{i-1}+1}^{k_i} iP(j) \tag{2.8}$$

while the expected number of workers in the back room is

$$B = N - F. \tag{2.9}$$

The expected number of customers in the front room is

$$L = \sum_{j=k_0}^{S} jP(j). \tag{2.10}$$

Expected waiting time in the queue can be expressed as

$$W_q = \frac{L}{\lambda(1-P(k_N))} - \frac{1}{\mu}. \tag{2.11}$$

This expression is derived using Little's Laws for a system of capacity $k_N = S$. Given a family of switching policies $\mathbf{K} = \{K; K = \{k_0, k_1, ..., k_{N-1}, S\}$, k_i integers, $k_i - k_{i-1} \geq 1$, $k_0 \geq 0, k_{N-1} < S\}$, the problem can formally be stated as:

$$minimize_{K\in\mathbf{K}}\ W_q \tag{2.12}$$

$$s.t.\ B \geq B_l$$

$$equations\ (2.1)\ to\ (2.4)$$

$$\textstyle\sum_{j=k_0}^{S} P(j) = 1$$

$$equations\ (2.8), (2.9), (2.10), (2.11).$$

It is important to note that B, F and L are expected values and can be real-valued. Consequently, the constraint $B \geq B_l$ states that the expected number of workers in the back room

resulting from the realization of any policy should be greater than or equal to the minimum expected number of back room workers needed to complete all work in the back room. At any particular time point, there may, in fact, be fewer than B_l workers in the back room.

2.4.1.2 Berman et al.'s Heuristic

Berman et al. [15] propose a heuristic method for the solution of this problem. This method is based on two corollaries and a theorem, which are stated and proved by the authors.

In particular, the major theorem from the paper of Berman et al. [15] is presented below as Theorem 2.4.1. For details and proof of this theorem, the reader is referred to Berman et al.'s work [15].

Theorem 2.4.1 (Berman et al.'s Theorem 1) *Consider two policies K and K' which are equal in all but one k_i. In particular, suppose that the value of k'_J equals $k_J - 1$, for some J from the set $\{0, ..., N-1\}$ such that $k_J - k_{J-1} \geq 2$, while $k'_i = k_i$ for all $i \neq J$. Then (a) $W_q(K) \geq W_q(K')$, (b) $F(K) \leq F(K')$, (c) $B(K) \geq B(K')$.*

As a result of Theorem 2.4.1, it can be seen that two policies exist which have special properties. Firstly, consider the policy

$$\hat{K} = \{k_0 = 0, k_1 = 1, k_2 = 2, ..., k_{N-1} = N-1, k_N = S\}.$$

This policy results in the largest possible F, and the smallest possible B and W_q. Because this policy yields the smallest possible expected waiting time, it is optimal if it is feasible. On the other hand, the smallest possible F and the largest possible W_q and B are obtained by applying the policy

$$\hat{\hat{K}} = \{k_0 = S-N, k_1 = S-N+1, ..., k_{N-1} = S-1, k_N = S\}.$$

Therefore, if this policy is infeasible, the problem (2.12) is infeasible also.

Berman et al. propose the notions of eligible type 1 and type 2 components. An *eligible type 1 component* is a switching point k_i satisfying the condition that $k_i - k_{i-1} > 1$ for $0 < i < N$

or $k_i > 0$ for $i = 0$. A switching point k_i is an *eligible type 2 component* if $k_{i+1} - k_i > 1$ for $0 \leq i < N$. More simply, an eligible type 1 component is a k_i variable which, if decreased by 1, will still be greater than k_{i-1}, while an eligible type 2 component is a k_i variable which, if increased by 1, will remain smaller than k_{i+1}. Eligible type 1 components and eligible type 2 components will further be referred to simply as type 1 and type 2 components, respectively.

Based on the definitions of policies $\hat{K}$ and $\hat{\hat{K}}$, the notions of type 1 and 2 components, and Theorem 2.4.1, Berman et al. [15] propose a heuristic, which has the same name as the problem it used for, P_1:

1. Start with $K = \hat{\hat{K}}$.

2. If $B(K) < B_l$, the problem is infeasible. Otherwise, let $imb\_W_q = W_q(K)$ and $imb\_K = K$. Set $J = N$.

3. Find the smallest j^* s.t. $0 \leq j^* < J$ and k_{j^*} is a type 1 component. If no such j^* exists, go to 5. Otherwise, set $k_{j^*} = k_{j^*} - 1$. If $B(K) < B_l$, set $J = j^*$ and go to 5. If $B(K) \geq B_l$, go to 4.

4. If $W_q(K) < imb\_W_q$, let $imb\_W_q = W_q(K)$ and $imb\_K = K$. Go to 3.

5. Find the smallest j^* s.t. $0 \leq j^* < J$ and k_{j^*} is a type 2 component. If no such j^* exists, go to 6. Otherwise, set $k_{j^*} = k_{j^*} + 1$. If $B(K) < B_l$, repeat 5. If $B(K) \geq B_l$, go to 4.

6. Stop and return imb_K as the best solution.

Limiting the choice of j^* to being between 0 and J, and resetting J every time an infeasible policy is found, prevents the heuristic from entering an infinite cycle. The heuristic guarantees optimality only when the policy it returns is $\hat{K}$ or $\hat{\hat{K}}$.

The ability of heuristic P_1 to find good switching policies is not explicitly evaluated in Berman et al.'s paper [15]. In particular, it is not clear how close the policies provided by P_1 are to the optimal policies.

2.4.1.3 Discussion

Problem P_1 is an example of a queue control problem that is very similar in nature to the problems discussed by Moder & Phillips [60], Hersh [44], Yadin & Naor [102] and Serfozo & Lu [74], which are mentioned in Section 2.2.2.

Moder & Phillips [60] assume that some workers are always available to serve the arriving customers, while others are able to perform tasks which are not directly related to the arrival process. Using the terminology of Berman et al., it can be said that some of the workers are cross-trained and able both to serve arriving customers in the front room and perform back room work. It is important to note that in the Moder & Phillips paper, unlike in the work of Berman et al. [15], there is no constraint on the amount of "back room work" that has to be performed, and some fixed number of workers are assumed to always be in the "front room". Moreover, the policies used in the two papers are quite different, although they are based on the same idea of increasing and decreasing the number of workers in the front room when the queue length reaches some threshold value. The policy used by Moder & Phillips is stated in terms of two parameters, v and N, so that the number of workers who serve customers is increased by 1 every time the queue length reaches N and decreased by 1 each time the queue length falls below v. The policy of Berman et al. is more flexible because it is expressed in terms of a larger number of parameters. It should also be noted that the Moder and Phillips paper derives a set of balance equations for calculating the probability of having a particular number of workers in the queue under the assumption of a (v, N) policy, which are very similar to the balance equations (2.1)–(2.4). Although expressions for quantities of interest such as the mean length of the queue are derived, and some numerical instances are examined by Moder & Phillips [60], no optimization problems are explicitly discussed.

Hersh [44] also assumes that a service facility has two types of work, "queue" work and "non-queue" work, which are the equivalent of Berman et al.'s front room and back room work. However, instead of placing a constraint on the amount of work that needs to be performed in the back room, he assumes that a cost is incurred for all of the non-queue work that is not

completed during the work day. A procedure based on simulation and dynamic programming is proposed for evaluation of different (v, N) policies and the determination of the optimal one.

The paper of Serfozo and Lu [74], unlike that of Berman et al. [15], deals with a queueing system which employs only one server. Moreover, it is concerned with the determination of optimal service and arrival rates rather than the number of servers. However, we note that the increase or decrease of workers in the front room considered in the work of Berman et al. [15] is equivalent to an increase or decrease of the total service rate. Consequently, using the approach of Serfozo and Lu, we can describe the problem of Berman et al. as that of determining the optimal service rate/arrival rate pair when there is a customer arrival. In particular, the arrival rate can be held constant, whereas the values of the service rate can be taken from the set $\{0, \mu, 2\mu, 3\mu, \ldots, N\mu\}$ corresponding to $0, 1, 2, 3, \ldots, N$ workers being present in the front room. The focus of the Serfozo paper is on the theoretical properties of their proposed policy rather than on the determination of the numerical optimal policy.

A similarity can be found between the policy type of Yadin et al. [102] and that of Berman et al. In particular, Yadin et al. use two vectors of queue lengths, one stating when the service rate should be switched to a higher value, and the other stating when it should be switched to a lower value (see Section 2.2.2). In Berman et al.'s formulation, one vector with the same structure is used to serve both purposes.

Overall, it can be seen that the Berman et al. paper can be considered as a logical extension of the most fundamental work in queueing control theory. However, it appears to be one of the rare papers dealing with the numerical optimization of a particular policy. Its assumption of a constraint on the back room work appears as reasonable as the assumption made by Hersh [44] of a cost being incurred for back room work completed in overtime. The formulation of the policy can be as easily used in practice as the simple (v, N) policy of Moder and Phillips [60], and provides almost as much flexibility and control as the policy of Yadin & Naor [102] that is based on two vectors rather than one.

Moreover, the problem as it is described in Berman et al. is of practical interest. Many

examples of facilities which could make use of the policies obtained from solving this problem exist in real life. For example, a grocery store employing workers who can re-stock shelves as well as serve customers at the check-out may be interested in finding out how to switch workers between the two tasks so as to minimize customer waiting time. More importantly, in a hospital, it may be necessary to determine how many nurses should be available to treat arriving emergency patients and how many should be treating patients who are already in the hospital.[3] Additionally, the problem could arise in computer science applications similar to those described in the paper of Palmer & Mitrani [63].

2.4.2 Berman et al.'s Problem P_2

Berman et al. [15] present a second problem, P_2. The goal of this problem is to find the minimum number of cross-trained workers, N, required in the retail facility subject to two constraints which ensure that the expected waiting time is bounded from above and that all work in the back room is performed.

Given a family of switching policies $\mathbf{K} = \{K; K = \{k_0, k_1, ..., k_{N-1}, S\},\ k_i$ integers, $k_i - k_{i-1} \geq 1, k_0 \geq 0, k_{N-1} < S\}$, the problem P_2 can be stated as:

$$
\begin{gathered}
minimize_{K \in \mathbf{K}}\ N \\
s.t.\ W_q \leq W_u \\
B \geq B_l \\
equations\ (2.1)\ to\ (2.4) \\
\textstyle\sum_{j=k_0}^{S} P(j) = 1 \\
equations\ (2.8), (2.9), (2.10), (2.11),
\end{gathered}
\tag{2.13}
$$

where W_u is the upper bound on the expected customer waiting time. Berman et al. propose a heuristic for solving problem P_2. This heuristic, which is referred to as heuristic P_2, enumerates

[3] This application of the back room/front room problem has been suggested by Dr. J. Hirdes.

N, starting at 1, and uses heuristic P_1 to try to find a feasible policy for the given N. P_2 stops as soon as P_1 is able to find a feasible policy. P_2 can be stated as follows:

1. Set $N = 1$.
2. Define policies $\hat{K}$ and $\hat{\hat{K}}$. Let $B_{min} = B(\hat{K})$, $W_{min} = W_q(\hat{K})$, $B_{max} = B(\hat{\hat{K}})$ and $W_{max} = W_q(\hat{\hat{K}})$.
3. If $B_{max} < B_l$ or $W_{min} > W_u$, no feasible policy exists for the current value of N–go to step 5. If $B_{min} \geq B_l$ and $W_{max} \leq W_u$, any policy $K \in \mathbf{K}$ (including $\hat{K}$ is optimal)–stop. Otherwise, go to step 4.
4. Apply Heuristic P_1. If Heuristic P_1 finds a policy, K^0, such that $B(K^0) \geq B_l$ and $W_q(K^0) \leq W_u$, stop and return K^0 and the current value of N.[4]
5. Set $N = N + 1$. Go to Step 2.

A set of 54 instances was used by Berman et al. [15] to evaluate the performance of this heuristic. In all instances, S is fixed to 100, and values of λ are taken from the set $\{10, 20, 30, 50, 75, 100\}$. Service rates μ are set to λ/R, where the value of R is 10, 15 or 20. R can be interpreted as the ratio of the arrival rate over the average service rate. B_l is calculated as a product of R and h, where h is set to 0.05, 0.1 or 0.2. The instances were generated using these parameters in a way that would ensure non-trivial solutions. For details regarding the instance generation procedure, see Berman et al.'s work [15].

The performance of P_2 was evaluated by comparison with a complete enumeration procedure. A time limit of one hour was placed on the run-times of both methods. The complete enumeration method was unable to find the optimal solution in 30 out of the 54 instances within the one-hour time limit. In the remaining 24 instances, the proposed heuristic was also able to find the optimal solution. In fact, the run-time of the heuristic, for each of the 54 instances

[4] In Berman et al.'s paper [15], it is stated that this N is "optimal". However, since heuristic P_1 may not be able to find a feasible policy for a particular N even if one exists, the value of N returned by heuristic P_2 may not be optimal.

tested, is smaller than 2 seconds. In a second experiment, 12 of the 30 instances which the complete enumeration procedure was unable to solve within one hour were adjusted, by decreasing the value of B_l, so that they could be solved within an hour by the complete method. For these instances, the heuristic was able to find the exact optimal value of N in all cases. The run-times of the heuristic were at most 1 second for any of these instances.

2.4.2.1 Discussion

Problem P_2 is a both a queue design and a queue control problem. It is a design problem because one needs to determine the optimal value of a parameter which will then be fixed–the total number of workers in the facility. It is a queue control problem because once the total number of workers to be employed is determined, one needs to effectively manage them so as to ensure that both types of work are completed at a satisfactory level.

In the design category, the problem is quite a bit different from the ones typically appearing in the literature. Namely, none of the papers discussed in Section 2.2.1 deal with determining an optimal parameter of the queue that would depend on an additional problem of finding a satisfactory control policy.

Similarly, the problem P_2 is quite different from the cross-training literature surveyed in Section 2.3, mainly because it is based on very specific assumptions about both types of work and allows an explicit statement of the balance equations. Unlike in the work of Brusco & Johns [20], and Cezik & L'Ecuyer [22], the notion of time is not explicitly taken into account in Berman et al.'s work. Moreover, Berman et al. assume that all workers are cross-trained, whereas the papers of Slomp et al. [76], Chevalier & Tabordon [23], and Cezik & L'Ecuyer [22] all assume the existence of several types of workers, some of whom may be able to perform more than two tasks. Motivated by the cross-training literature, in Chapter 4 we relax the assumption of Berman et al. that only cross-trained workers are employed in the facility.

2.5 Conclusion

Problems P_1 and P_2 of Berman et al. are problems of potential practical interest dealing with optimal design and control of a queue. Berman et al. have devised efficient heuristics for these problems, which, however, do not guarantee optimality. In fact, the quality of the solution these methods provide is explictly evaluated by Berman et al. only on a small set of instances and only for problem P_2.

In Section 2.1, we introduced constraint programming, an approach that has been successfully used to solve many combinatorial problems [72]. We also presented a brief survey of how constraint programming has been extended to solve problems which involve uncertainty and change. Although a variety of such extensions exist, the lack of integration of constraint programming with areas of operations research dealing with stochastic or dynamic problems, such as queueing theory, has been noted by Brown & Miguel [19].

In order to attempt to provide a link between constraint programming and queueing theory, in Chapter 3 we apply constraint programming to Berman et al.'s problem P_1. We reinforce this link by showing, in Chapter 4, that a CP-based method may be used to solve an extension of P_2 inspired by the cross-training literature. A secondary goal of this work is to propose complete, rather than heuristic, methods for solving these problems.

Chapter 3

Constraint Programming for a Queue Control Problem

As indicated in preceding chapter, the theory and methods of constraint programming have been extended to solving problems involving uncertainty and change. However, to our knowledge, no one has attempted to apply CP to a stochastic queueing control problem. In this chapter, we investigate whether CP can be successfully used to solve a particular queueing control problem, namely, problem P_1 of Berman et al., discussed in Section 2.4.[1]

In the following section, three CP models for problem P_1 are proposed. Sections 3.2 and 3.3 present methods for improving the efficiency of these models, focusing on dominance rules and shaving procedures, respectively. Section 3.4 shows experimental results comparing the proposed CP models and combinations of inference methods. The performance of the CP techniques is contrasted with that of the heuristic method of Berman et al. Based on these results, a hybrid method is proposed and evaluated in Section 3.5. In Section 3.6, a discussion of the results is presented. Section 3.7 concludes this chapter.

[1] A shorter version of this chapter appears as [93].

3.1 Constraint Programming Models

Berman et al.'s problem P_1 concerns a retail facility of capacity S which has back room and front room operations and employs N cross-trained workers. The goal of the problem is to find a policy which states how these workers should be switched between the two rooms. More specifically, we seek a switching policy which minimizes the expected customer waiting time, W_q, but ensures that the expected number of workers in the back room, B, remains greater than or equal to the minimum expected number of workers, B_l, necessary in the back room for completion of all back room tasks.[2] In the front room, customers arrive according to a Poisson process with rate λ, and service times follow an exponential distribution with rate μ.

We investigate three CP models for this problem:

- The *If-Then* model is a CP version of the formal definition of Berman et al.
- The *PSums* model uses a slightly different set of variables, and most of the constraints are based on closed-form expressions derived from the constraints that are used in the *If-Then* model.
- The *Dual* model includes a set of dual decision variables in addition to the variables used in the *If-Then* and *PSums* models. Most of the constraints of this model are expressed in terms of these dual variables.

The proposed models have some similarities. Recall from Section 2.4 that a policy is represented by the set $\{k_i, i = 0, 1, \ldots, N\}$, with the interpretation that whenever the number of customers in the front room is between $k_{i-1} + 1$ and k_i, the number of workers in the front room is i. Consequently, all three of the proposed models have a set of decision variables k_i, $i = 0, 1, \ldots, N$, representing the switching policy. Each k_i from this set has the domain $[i, i + 1, \ldots, S - N + i]$ and has to satisfy the constraint $k_i < k_{i+1}$ (since the number of work-

[2] Following Berman et al. [15], we assume only one policy *type*. In this context, an *optimal policy* is a set of numerical values for the parameters defining this policy type which results in the lowest possible W_q subject to the back room constraint.

ers in the front room, i, increases only when the number of customers, k_i, increases). Due to the policy definition of Berman et al., k_N is constrained to equal to S.

All models include variables and constraints for the representation of the balance equations (Equations (2.1)–(2.4)), and expressions for F, the expected number of workers in the front room, and L, the expected number of customers in the front room. However, these representations differ slightly depending on the model, as noted below in Sections 3.1.1, 3.1.2 and 3.1.3. The equation for W_q is exactly the same in all models and is stated as Equation (3.1), where $P(k_N)$ is the probability of having $k_N = S$ customers in the front room. This equation is taken directly from the formulation of Berman et al. and has previously been stated in Equation (2.11).

$$W_q = \frac{L}{\lambda(1 - P(k_N))} - \frac{1}{\mu} \tag{3.1}$$

A set of auxiliary variables $\beta Sum(k_i)$, for all i from 1 to $N - 1$, is included in each of the models. These are necessary for representing Equation (3.2), which relates these variables to $P(k_0)$, a floating point variable with domain $[0..1]$ representing the probability of having k_0 customers in the facility. These auxiliary variables and constraint ensure that an assignment of all decision variables leads to a unique solution of the balance equations. We discuss the formal definition of these auxiliary variables in Section 3.1.4.

The back room constraint, $B \geq B_l$, is stated in all models as $N - F \geq B_l$.

$$P(k_0) \sum_{i=0}^{N} \beta Sum(k_i) = 1 \tag{3.2}$$

3.1.1 *If-Then* Model

The initial model is based directly on the formulation of Berman et al. and therefore includes the variables $P(j)$ for $j = k_0, k_0 + 1, \ldots, k_1, k_1 + 1, \ldots, k_N - 1, k_N$, each representing the steady-state probability of there being j customers in the front room. These floating point

variables with domain $[0..1]$ have to satisfy a system of balance equations (Equations (2.1)–(2.4)) and are used to express L and F.

3.1.1.1 Balance Equation Constraints

In this model, balance equations are represented by a set of if-then constraints. For example, the first balance equation, $P(j)\lambda = P(j+1)\mu$ for $j = k_0, k_0+1, ..., k_1-1$, is represented by the constraint $(k_0 \leq j \leq k_1 - 1) \rightarrow P(j)\lambda = P(j+1)\mu$. Thus, somewhat inelegantly, an if-then constraint of this kind has to be added for each j between 0 and $S-1$ (inclusive) in order to represent one balance equation. In order to represent the rest of these equations, this technique has to be applied for each pair of switching points k_i, k_{i+1} for i from 0 to $N-1$. This results in a total of $N \times S$ if-then constraints.

The probabilities $P(j)$ also have to satisfy the constraint $\sum_{j=k_0}^{S} P(j) = 1$. The difficulty with this constraint is the fact that the sum starts at $j = k_0$, where k_0 is a variable. In order to deal with this complication, we add the meta-constraint (refer to Section 2.1.2 for a definition) $((j < k_0) \leq (P(j) = 0))$ for each j from the set $\{0, 1, \ldots, S-N-1\}$.[3] This implies that all values of $P(j)$ with j less than k_0 will be 0 and allows us to express the sum-of-probabilities constraint as $\sum_{j=0}^{S} P(j) = 1$.

The constraint necessary for ensuring a unique solution to the balance equations once all decision variables have been assigned, which is stated in Equation (3.2), requires $S+1$ if-then constraints. Each of these constraints has the following form:
$(k_0 = j) \rightarrow P(j) \sum_{i=0}^{N} \beta Sum(k_i) = 1$ (refer to Section 3.1.4 for details regarding the calculation of $\sum_{i=0}^{N} \beta Sum(k_i)$).

[3] We do not need to add this constraint for all j from 0 to S because the upper bound of the domain of k_0 is $S-N$.

3.1.1.2 Expected Number of Workers Constraints

A set of if-then constraints also has to be included in order to represent Equation (3.3) (restatement of Equation (2.8)) as a constraint in our model. This is due to the dependence of this constraint on sums of variables between two switching points, which are also variables.

$$F = \sum_{i=1}^{N} \sum_{j=k_{i-1}+1}^{k_i} iP(j) \tag{3.3}$$

More specifically, we add a set of variables $product(i, j)$ for representing the product of i and $P(j)$ when j is between $k_{i-1} + 1$ and k_i, and the constraints $(k_{i-1} + 1 \leq j \leq k_i) \rightarrow product(i, j) = iP(j)$ and $(k_{i-1} + 1 > j \ \ || \ \ j > k_i) \rightarrow product(i, j) = 0$ for all i from 1 to N and for all j from 0 to S. The total number of these if-then constraints is $2N(S + 1)$. F can then be simply stated as a sum over the indices i and j of variables $product(i, j)$.

3.1.1.3 Expected Number of Customers Constraint

L is defined according to Equation (2.10). Since the meta-constraint $((j < k_0) \leq (P(j) = 0))$ has been added to the model in order to ensure that $P(j) = 0$ for all $j < k_0$, the constraint for L can be simply stated as the sum of the products of j and $P(j)$ over all j from 0 to S:

$$L = \sum_{j=0}^{S} jP(j). \tag{3.4}$$

3.1.1.4 Overall Model

The complete *If-Then* model can therefore be stated as:

$$
\begin{aligned}
minimize \;\; & W_q \\
subject \;\; to & \\
k_i &< k_{i+1} && \forall i \in \{0,1,\ldots,N-1\}; \\
k_N &= S; \\
(k_i \le j \le k_{i+1}-1) &\rightarrow P(j)\lambda = P(j+1)(i+1)\mu, \\
& \quad \forall i \in \{0,1,\ldots,N-1\}, \forall j \in \{0,1,\ldots,S-1\}; \\
(j < k_0) &\le (P(j)=0), && \forall j \in \{0,1,\ldots,S-N-1\}; \\
\sum_{j=0}^{S} P(j) &= 1; \\
(k_0 = j) &\rightarrow P(j)\sum_{i=0}^{N} \beta Sum(k_i) = 1, \\
& && \forall j \in \{0,1,\ldots,S\}; \\
L &= \sum_{j=0}^{S} jP(j); \\
(k_{i-1}+1 \le j \le k_i) &\rightarrow product(i,j) = iP(j), \\
& \quad \forall i \in \{1,2,\ldots,N\}, \forall j \in \{0,1,\ldots,S\}; \\
(k_{i-1}+1 > j \;\; || \;\; j > k_i) &\rightarrow product(i,j) = 0, \\
& \quad \forall i \in \{1,2,\ldots,N\}, \forall j \in \{0,1,\ldots,S\}; \\
F &= \sum_{i=1}^{N}\sum_{j=0}^{S} product(i,j); \\
W_q &= \frac{L}{\lambda(1-P(k_N))} - \frac{1}{\mu}; \\
N - F &\ge B_l; \\
auxiliary & \quad constraints,
\end{aligned}
$$

where $auxiliary\ constraints$ are constraints necessary for defining $\beta Sum(k_i)$ and are discussed in Section 3.1.4.

This model includes a total of $3NS+2N+S$ if-then constraints, which are highly inefficient because propagation only occurs either when the left-hand side is satisfied or when the right-hand side becomes false. Consequently, the next model attempts to avoid, as much as possible, the use of such constraints.

3.1.2 *PSums* Model

The second constraint programming model is based on closed-form expressions that are derived from the balance equations (Equations (2.1)–(2.4)). The set of $P(j)$ variables from the formulation of Berman et al. is replaced by a set of $PSums(k_i)$ variables for $i = 0, \ldots, N-1$, together with a set of probabilities $P(j)$ for $j = k_0, k_1, k_2, \ldots, k_N$. Note that $P(j)$ is defined for each switching point only, not for all values from $\{0, 1, \ldots, S\}$. The $PSums(k_i)$ variable represents the sum of all probabilities between k_i and $k_{i+1} - 1$.

3.1.2.1 Probability Constraints

Balance equations are not explicitly stated in this model. However, expressions for $P(k_i)$ and $PSums(k_i)$ are derived in such a way that the balance equations are satisfied. The technique used for these derivations is similar to that used by Berman et al. [15] to simplify the calculation of probabilities.

Consider the balance equation $P(j)\lambda = P(j+1)i\mu$, which is true for $j = k_{i-1}, k_{i-1} + 1, \ldots, k_i - 1$ and any $i \in \{1, 2 \ldots, N\}$. In particular, this subset of the balance equations is

$$\begin{aligned} P(k_{i-1})\lambda &= P(k_{i-1}+1)i\mu \\ P(k_{i-1}+1)\lambda &= P(k_{i-1}+2)i\mu \\ &\vdots \\ P(k_i-1)\lambda &= P(k_i)i\mu. \end{aligned}$$

These equations imply the following expressions:

$$\begin{aligned} \frac{P(k_{i-1})\lambda}{i\mu} &= P(k_{i-1}+1) \\ \frac{P(k_{i-1}+1)\lambda}{i\mu} &= P(k_{i-1}+2) \\ &\vdots \\ \frac{P(k_i-1)\lambda}{i\mu} &= P(k_i). \end{aligned} \tag{3.5}$$

Combining these together, we get $P(k_i) = \left(\frac{\lambda}{i\mu}\right)^{k_i - k_{i-1}} P(k_{i-1})$ for all i from 1 to N, or

$$P(k_{i+1}) = \left(\frac{\lambda}{(i+1)\mu}\right)^{k_{i+1}-k_i} P(k_i), \quad \forall i \in \{0, 1, \ldots, N-1\}. \tag{3.6}$$

Consequently, Equation (3.6) is a recursive formula for computing $P(k_{i+1})$. $P(k_0)$ can be computed by using Equation (3.2). As in the other two models, the implementation of this constraint requires the use of auxiliary variables, $\beta Sum(k_i)$, which are discussed in detail in Section 3.1.4. However, unlike in the other two models, this constraint does not require any if-then constraints since variable $P(k_0)$ is explicitly defined.

Similarly, from Equation (3.5), we see that $P(k_{i-1}+1) = \frac{\lambda}{i\mu} P(k_{i-1})$ for all i from 1 to N, or

$$P(k_i + 1) = \frac{\lambda}{(i+1)\mu} P(k_i), \quad \forall i \in \{0, 1, \ldots, N\}. \tag{3.7}$$

Using Equation (3.7), an expression for $PSums(k_i)$, the sum of probabilities $P(j)$ for j between k_i and $k_{i+1} - 1$, can be derived as follows:

$$\begin{aligned}
PSums(k_i) &= \sum_{j=k_i}^{k_{i+1}-1} P(j) \\
&= P(k_i) + P(k_i+1) + P(k_i+2) + \cdots + P(k_{i+1}-1) \\
&= P(k_i) + P(k_i)\frac{\lambda}{(i+1)\mu} + P(k_i)\left(\frac{\lambda}{(i+1)\mu}\right)^2 \\
&\quad + \cdots + P(k_i)\left(\frac{\lambda}{(i+1)\mu}\right)^{k_{i+1}-k_i-1} \\
&= P(k_i)\left[1 + \frac{\lambda}{(i+1)\mu} + \left(\frac{\lambda}{(i+1)\mu}\right)^2 + \cdots + \left(\frac{\lambda}{(i+1)\mu}\right)^{k_{i+1}-k_i-1}\right] \\
&= \begin{cases} P(k_i)\dfrac{1 - \left[\dfrac{\lambda}{(i+1)\mu}\right]^{k_{i+1}-k_i}}{1 - \dfrac{\lambda}{(i+1)\mu}} & \text{if } \frac{\lambda}{(i+1)\mu} \neq 1 \\ P(k_i)(k_{i+1} - k_i) & otherwise. \end{cases}
\end{aligned} \tag{3.8}$$

The last step in the derivation is based on the observation that the expression $[1 + \frac{\lambda}{(i+1)\mu} + \left(\frac{\lambda}{(i+1)\mu}\right)^2 + \cdots + \left(\frac{\lambda}{(i+1)\mu}\right)^{k_{i+1}-k_i-1}]$ is a geometric series with the common ratio $\frac{\lambda}{(i+1)\mu}$. When $\frac{\lambda}{(i+1)\mu}$ is 1, the expression is simply a sum of $k_{i+1} - k_i$ ones.

Additionally, the probabilities have to satisfy the constraint $\sum_{i=0}^{N-1} PSums(k_i) + P(k_N) = 1$.

3.1.2.2 Expected Number of Workers Constraint

F can be expressed in terms of $P(k_i)$ and $PSums(k_i)$ using the following sequence of steps:

$$\begin{aligned} F &= \sum_{i=1}^{N} \sum_{j=k_{i-1}+1}^{k_i} iP(j) \\ &= \sum_{i=1}^{N} i\,[P(k_{i-1}+1) + P(k_{i-1}+2) + \cdots + P(k_i - 1) + P(k_i)] \\ &= \sum_{i=1}^{N} i\,[PSums(k_{i-1}) - P(k_{i-1}) + P(k_i)]. \end{aligned} \tag{3.9}$$

3.1.2.3 Expected Number of Customers Constraints

The equation for L can be derived as in a similar manner:

$$\begin{aligned} L &= \sum_{j=k_0}^{k_N} jP(j) \\ &= \sum_{j=k_0}^{k_1-1} jP(j) + \sum_{j=k_1}^{k_2-1} jP(j) + \cdots + \sum_{j=k_{N-1}}^{k_N-1} jP(j) + k_N P(k_N) \\ &= L(k_0) + L(k_1) + \cdots + L(k_{N-1}) + k_N P(k_N) \\ &= \sum_{i=0}^{N-1} L(k_i) + k_N P(k_N) \end{aligned} \tag{3.10}$$

where

$$
\begin{aligned}
L(k_i) &= k_iP(k_i) + (k_i+1)P(k_i+1) + (k_i+2)P(k_i+2) + \cdots + (k_{i+1}-1)P(k_{i+1}-1) \\
&= k_iP(k_i) + k_iP(k_i+1) + k_iP(k_i+2) + \cdots + k_iP(k_{i+1}-1) + P(k_i+1) \\
&+ 2P(k_i+2) + \cdots + (k_{i+1}-k_i-1)P(k_{i+1}-1) \\
&= k_i[P(k_i) + P(k_i+1) + P(k_i+2) + \cdots + P(k_{i+1}-1)] + P(k_i+1) \\
&+ 2P(k_i+2) + \cdots + (k_{i+1}-k_i-1)P(k_{i+1}-1) \\
&= k_iPSums(k_i) + P(k_i)\frac{\lambda}{(i+1)\mu} + 2P(k_i)\left(\frac{\lambda}{(i+1)\mu}\right)^2 \\
&+ \cdots + (k_{i+1}-k_i+1)P(k_i)\left(\frac{\lambda}{(i+1)\mu}\right)^{k_{i+1}-k_i-1} \\
&= k_iPSums(k_i) + P(k_i)\frac{\lambda}{(i+1)\mu} \times \left[1 + 2\frac{\lambda}{(i+1)\mu} + 3\left(\frac{\lambda}{(i+1)\mu}\right)^2 + \ldots \right. \\
&+ \left. (k_{i+1}-k_i-1)P(k_i)\left(\frac{\lambda}{(i+1)\mu}\right)^{k_{i+1}-k_i-2}\right] \\
&= k_iPSums(k_i) + P(k_i)\frac{\lambda}{(i+1)\mu} \sum_{n=0}^{k_{i+1}-k_i-1} n\left(\frac{\lambda}{(i+1)\mu}\right)^{n-1} \\
&= k_iPSums(k_i) + P(k_i)\frac{\lambda}{(i+1)\mu} \\
&\quad \times \left[\frac{\left(\frac{\lambda}{(i+1)\mu}\right)^{k_{i+1}-k_i-1}(k_i-k_{i+1}) + \left(\frac{\lambda}{(i+1)\mu}\right)^{k_{i+1}-k_i}(k_{i+1}-k_i-1)+1}{\left(1-\frac{\lambda}{(i+1)\mu}\right)^2}\right].
\end{aligned}
$$

3.1.2.4 Overall Model

The *PSums* model can be summarized as follows:

$$
\begin{aligned}
&minimize && W_q \\
&subject\ to && \\
k_i &< k_{i+1} && \forall i \in \{0,1,\ldots,N-1\}; \\
k_N &= S; && \\
P(k_{i+1}) &= \left(\frac{\lambda}{(i+1)\mu}\right)^{k_{i+1}-k_i} P(k_i), && \\
& && \forall i \in \{0,1,\ldots,N-1\}; \\
PSums(k_i) &= \begin{cases} P(k_i)\dfrac{1-\left[\dfrac{\lambda}{(i+1)\mu}\right]^{k_{i+1}-k_i}}{1-\dfrac{\lambda}{(i+1)\mu}} & \text{if } \frac{\lambda}{(i+1)\mu} \neq 1 \\ P(k_i)(k_{i+1}-k_i) & otherwise \end{cases}, && \\
& && \forall i \in \{1,2,\ldots,N-1\}; \\
\sum_{i=0}^{N-1} PSums(k_i) &+ P(k_N) = 1; && \\
P(k_0) &\times \sum_{i=0}^{N} \beta Sum(k_i) = 1; && \\
F &= \sum_{i=1}^{N} i\left[PSums(k_{i-1}) - P(k_{i-1}) + P(k_i)\right]; && \\
L &= \sum_{i=0}^{N-1} L(k_i) + k_N P(k_N), && \\
L(k_i) &= k_i PSums(k_i) + P(k_i)\frac{\lambda}{(i+1)\mu} && \\
&\times \frac{\left(\frac{\lambda}{(i+1)\mu}\right)^{k_{i+1}-k_i-1}(k_i-k_{i+1}) + \left(\frac{\lambda}{(i+1)\mu}\right)^{k_{i+1}-k_i}(k_{i+1}-k_i-1)+1}{\left(1-\frac{\lambda}{(i+1)\mu}\right)^2}, && \\
& && \forall i \in \{0,1,\ldots,N-1\}; \\
W_q &= \frac{L}{\lambda(1-P(k_N))} - \frac{1}{\mu}; && \\
N-F &\geq B_l; && \\
auxiliary & \ constraints, &&
\end{aligned}
$$

where $auxiliary\ constraints$ are constraints necessary for calculating the values of $\beta Sum(k_i)$.

These are discussed in Section 3.1.4.

3.1.3 *Dual* Model

The problem can be alternatively formulated using variables w_j, which represent the number of workers in the front room when there are j customers present. The w_j variables can be referred to as the *dual* variables because, compared to the k_is, the roles of variables and values are switched [48, 82].

In our *Dual* model, there are $S+1$ w_j variables, each with domain $[0, 1, \ldots, N]$. These variables have to satisfy the following equations: $w_0 = 0$, $w_S = N$ and $w_j \leq w_{j+1}$ for all j from 0 to $S-1$. Additionally, the complete set of k_i variables is included in this model, since some constraints are easier to express using the k_is rather than the w_js.

3.1.3.1 Probability Constraints

Given the set of dual variables, the balance equations can be restated as

$$P(j)\lambda \;=\; P(j+1)w_{j+1}\mu, \quad \forall j \in \{0, 1, \ldots, S-1\}. \tag{3.11}$$

This formulation of the balance equations avoids the inefficient if-then constraints. The rest of the restrictions on the probability variables are stated in terms of the k_i variables, as in the *If-Then* model. In particular, the constraints $\sum_{j=0}^{S} P(j) = 1$, $((k_0 > j) \leq (P(j) = 0))$ $\forall j \in \{0, \ldots, S-N-1\}$, and $(k_0 = j) \rightarrow P(j) \sum_{i=0}^{N} \beta Sum(k_i) = 1$ $\forall j \in \{0, 1, \ldots, S\}$ (refer to Section 3.1.4 for details regarding the calculation of $\sum_{i=0}^{N} \beta Sum(k_i)$) are present in this model.

3.1.3.2 Channelling Constraints

In order to use redundant variables, a set of channelling constraints has to be added to the model to ensure that an assignment of values to one set of variables will lead to a unique assignment

of variables in the other set. The following channelling constraints[4] are included:

$$w_j < w_{j+1} \quad \leftrightarrow \quad k_{w_j} = j \qquad \forall j \in \{0, 1, \ldots, S-1\}, \tag{3.12}$$

$$w_j = w_{j+1} \quad \leftrightarrow \quad k_{w_j} \neq j \qquad \forall j \in \{0, 1, \ldots, S-1\}, \tag{3.13}$$

$$w_j = i \quad \leftrightarrow \quad k_{i-1} + 1 \leq j \leq k_i \qquad \forall j \in \{0, 1, \ldots, S\}, \;\; \forall i \in \{1, \ldots, N\}. \tag{3.14}$$

3.1.3.3 Expected Number of Workers Constraint

The expression for the expected number of workers in the front room is $F = \sum_{j=0}^{S} w_j P(j)$.

3.1.3.4 Expected Number of Customers Constraint

The constraint used to express L is identical to the one used in the *If-Then* model:

$$L \;\; = \;\; \sum_{j=0}^{S} jP(j).$$

This equation is valid because all $P(j)$ for $j < k_0$ are constrained to be 0.

[4]Constraints (3.12) and (3.13) are redundant given the constraint $w_j \leq w_{j+1}$. However, such redundancy can often lead to increased propagation [48]. One direction for future work is examining the effect that removing one of these constraints may have on the performance of the program.

3.1.3.5 Overall Model

The *Dual* model can be stated as:

$$
\begin{array}{rcll}
minimize & W_q & & \\
subject\ to & & & \\
w_0 = 0, & & w_S = N & \\
w_j & \leq & w_{j+1} & \forall j \in \{0,1,\ldots,S-1\}; \\
k_i & < & k_{i+1} & \forall i \in \{0,1,\ldots,N-1\}; \\
k_N & = & S; & \\
w_j < w_{j+1} & \leftrightarrow & k_{w_j} = j & \forall j \in \{0,1,\ldots,S-1\}; \\
w_j = w_{j+1} & \leftrightarrow & k_{w_j} \neq j & \forall j \in \{0,1,\ldots,S-1\}; \\
w_j = i & \leftrightarrow & k_{i-1} + 1 \leq j \leq k_i & \forall i \in \{1,2,\ldots,N\}, \forall j \in \{0,1,\ldots,S\}; \\
P(j)\lambda & = & P(j+1)w_{j+1}\mu & \forall j \in \{0,1,\ldots,S-1\}; \\
(k_0 = j) & \rightarrow & P(j)\sum_{i=0}^{N} \beta Sum(k_i) = 1, & \forall j \in \{0,1,\ldots,S\}; \\
(j < k_0) & \leq & (P(j) = 0), & \forall j \in \{0,1,\ldots,S-N-1\}; \\
\sum_{j=0}^{S} P(j) & = & 1; & \\
F & = & \sum_{j=0}^{S} w_j P(j); & \\
L & = & \sum_{j=0}^{S} jP(j); & \\
W_q & = & \dfrac{L}{\lambda(1-P(k_N))} - \dfrac{1}{\mu}; & \\
N - F & \geq & B_l; & \\
auxiliary & & constraints. &
\end{array}
$$

$Auxiliary\ constraints$ are necessary for calculating $\beta Sum(k_i)$ and are discussed in the following section.

3.1.4 Auxiliary Variables and Constraints

Berman et al. derive, from the balance equations, the equation $P(k_0)\sum_{j=k_0}^{k_N}\beta_j = 1$ where

$$\beta_j = \begin{cases} 1 & \text{if} \quad j = k_0 \\ \left(\frac{\lambda}{\mu}\right)^{j-k_0}\left(\frac{1}{i}\right)^{j-k_{i-1}}X_i & \text{if} \quad k_{i-1}+1 \le j \le k_i \quad i = 1,\ldots,N \end{cases} \tag{3.15}$$

and

$$X_i \;=\; \prod_{g=1}^{i-1}\left(\frac{1}{g}\right)^{k_g-k_{g-1}}, \qquad (X_1 \equiv 1), i = 1,\ldots,N. \tag{3.16}$$

These expressions have to be included in each of the models in order to ensure that the values assigned to the probability variables, $P(j)$, or $PSums(k_i)$ and $P(k_i)$, constitute a unique solution to the balance equations (2.1)–(2.4).

Consequently, in all models, there is a set of $N-2$ auxiliary expressions X_i for all i from 3 to N (X_1 and X_2 are always equal to 1). There are also $N+1$ continuous auxiliary variables $\beta Sum(k_i)$ with domain $[0\ldots 1+S\left(\frac{\lambda}{\mu}\right)^S]$ which represent sums of β_j from $j = k_{i-1}+1$ to $j = k_i$. $\beta Sum(k_0)$ is constrained to equal 1, while the rest of these variables are defined according to Equation (3.17). The validity of this equation is proven by the following derivation, which

uses the formula for the sum of a finite geometric series in the last step:

$$\begin{aligned}
\beta Sum(k_i) &= \sum_{j=k_{i-1}+1}^{k_i} \beta_j \\
&= \left(\frac{\lambda}{\mu}\right)^{k_{i-1}+1-k_0} \left(\frac{1}{i}\right)^{k_{i-1}+1-k_{i-1}} X_i \\
&+ \left(\frac{\lambda}{\mu}\right)^{k_{i-1}+2-k_0} \left(\frac{1}{i}\right)^{k_{i-1}+2-k_{i-1}} X_i + \cdots + \left(\frac{\lambda}{\mu}\right)^{k_i-k_0} \left(\frac{1}{i}\right)^{k_i-k_{i-1}} X_i \\
&= X_i \left(\frac{\lambda}{\mu}\right)^{k_{i-1}-k_0+1} \left(\frac{1}{i}\right) \\
&\times \left[1 + \frac{\lambda}{\mu}\frac{1}{i} + \left(\frac{\lambda}{\mu}\right)^2 \left(\frac{1}{i}\right)^2 + \cdots + \left(\frac{\lambda}{\mu}\right)^{k_1-k_0-(k_{i-1}-k_0+1)} \left(\frac{1}{i}\right)^{k_i-k_{i-1}-1}\right] \\
&= X_i \left(\frac{\lambda}{\mu}\right)^{k_{i-1}-k_0+1} \left(\frac{1}{i}\right) \sum_{n=0}^{k_i-k_{i-1}-1} \left(\frac{\lambda}{i\mu}\right)^n
\end{aligned}$$

$$= \begin{cases} X_i \left(\frac{\lambda}{\mu}\right)^{k_{i-1}-k_0+1} \left(\frac{1}{i}\right) \left[\frac{1-\left(\frac{\lambda}{i\mu}\right)^{k_i-k_{i-1}}}{1-\left(\frac{\lambda}{i\mu}\right)}\right] & \text{if } \frac{\lambda}{i\mu} \neq 1 \\ X_i \left(\frac{\lambda}{\mu}\right)^{k_{i-1}-k_0+1} \left(\frac{1}{i}\right) (k_i - k_{i-1}) \quad otherwise. \end{cases} \tag{3.17}$$

The sum $\sum_{j=k_0}^{k_N} \beta_j$ can then be expressed as $\sum_{i=0}^{N} \beta Sum(k_i)$. The requirement that $P(k_0) \times \sum_{j=k_0}^{k_N} \beta_j$ has to equal 1 is stated in the *If-Then* model and in the *Dual* model as a set of if-then constraints $(k_0 = j) \rightarrow P(j) \sum_{i=0}^{N} \beta Sum(k_i) = 1$ for all $j \in \{0, 1, \ldots, S\}$. In the *PSums* model, this requirement is stated directly as $P(k_0) \sum_{i=0}^{N} \beta Sum(k_i)$, because this model has an explicit closed-form expression for the variable $P(k_0)$.

In summary, the $auxiliary\ constraints$ that are present in each of the models are:

$$\beta Sum(k_0) = 1,$$

$$X_i = \prod_{g=1}^{i-1} \left(\frac{1}{g}\right)^{k_g-k_{g-1}} \qquad \forall i \in \{3, \ldots, N\},$$

$$\beta Sum(k_i) \;=\; \begin{cases} X_i\left(\frac{\lambda}{\mu}\right)^{k_{i-1}-k_0+1}\left(\frac{1}{i}\right)\left[\frac{1-\left(\frac{\lambda}{i\mu}\right)^{k_i-k_{i-1}}}{1-\left(\frac{\lambda}{i\mu}\right)}\right] & \text{if } \frac{\lambda}{i\mu} \neq 1 \\ X_i\left(\frac{\lambda}{\mu}\right)^{k_{i-1}-k_0+1}\left(\frac{1}{i}\right)(k_i-k_{i-1}) \quad otherwise. & \end{cases}$$
$$\forall i \in \{1,\ldots,N\};$$

3.1.5 Summary

Table 3.1 presents a summary of the number of variables and constraints in each of the three proposed models. It can be seen that the *PSums* model has a smaller number of probability variables and constraints but a slightly larger number of constraints for representing L than the other two models, and no if-then constraints. The *Dual* model has a larger number of decision variables than the *If-Then* and *PSums* models. This does not imply that the search space is bigger in this model because the two sets of variables are linked by channelling constraints and so are assigned values via propagation. The *Dual* allows the simplest representations of F and L, each requiring only one constraint. The number of probability constraints in the *Dual* is smaller than or equal to the number of such constraints in the *If-Then* model and greater than in the *PSums* model. However, the actual representation of these constraints is the most straightforward in the *Dual* since it neither requires if-then constraints nor closed-form expressions. The *If-Then* model has, by far, the greatest number of constraints in total. Its only advantage over the *PSums* model is the significantly simpler representation of L.

It is hard to determine, simply by looking at the statistics presented in Table 3.1, which of the models will be more efficient since, in CP, a larger number of constraints and/or variables may actually lead to more propagation and to a more effective model [82]. However, it is known that if-then constraints do not propagate well, and, since the difference in the number of these constraints between the *PSums* model and both the *If-Then* model and the *Dual* model is quite significant, one may expect the *PSums* model to have some advantage over the other

Statistic/Model	*If-Then*	*PSums*	*Dual*
# of decision variables	$N+1$	$N+1$	$N+S+2$
# of probability variables	$S+1$	$2N+1$	$S+1$
# of probability constraints	$N(S-1)+2S+2$	$2N+1$	$3S-N+2$
# of constraints for F	$2N(S+1)+1$	1	1
# of constraints for L	1	$N+1$	1
# of if-then constraints	$3NS+2N+S$	0	$S+1$

Table 3.1: Summary of the main characteristics of the three proposed CP models.

two models.

In fact, preliminary experiments with all three models showed poor performance (see Table 3.2). Due to the small number of constraints between decision variables and the complexity of constraints relating the decision variables to variables representing probabilities, there was little constraint propagation, and, essentially, search was required to explore the entire branch-and-bound tree. As a consequence, in the following two sections we examine dominance rules and shaving, two stronger inference forms used in CP (introduced in Section 2.1.1.3). In Section 3.6, we investigate why the models without dominance rules and shaving need to search the whole tree in order to prove optimality, and also discuss differences in the performance of the models based on our experimental results.

3.2 Dominance Rules

A dominance rule is a constraint that forbids assignments of values to variables which are known to be sub-optimal. For this particular problem, a dominance rule can be derived by first noticing that any policy can be decomposed into two parts. The first part, which will be further referred to as the "head" of the policy, is an ordered sequence of k_is, starting from k_0, which are assigned their smallest possible values. The second part, considered as the "tail" of the policy,

contains all the rest of the switching points. The dominance rule that can be added to any of the proposed constraint programming models states that, given a feasible solution, K, all further solutions have to have at least one switching point in the tail of the policy assigned a lower value than the value assigned to it in K. In other words, given two solutions K and K', if the W_q value resulting from policy K' is smaller than or equal to the W_q value resulting from K, then there have to exist switching points k'_i and k_i in the tail of K' and K, respectively, satisfying the condition $k'_i < k_i$. The following theorem states the dominance rule more formally.

Theorem 3.2.1 (Dominance Rule) *Let $K = (k_0, k_1, \ldots, k_N)$ and $K' = (k'_0, k'_1, \ldots, k'_N)$ be two policies such that $k_0 = k'_0 = 0$, $k_1 = k'_1 = 1, \ldots, k_{J-1} = k'_{J-1} = J-1$ and $k_J \neq k'_J$ (i.e. at least one of k_J, k'_J is strictly greater than J) for some $J \in \{0, 1, \ldots, N-1\}$. Let $W_q(K)$ and $W_q(K')$ denote the expected waiting times resulting from the two policies K and K', respectively. If $W_q(K') < W_q(K)$, then there exists $i \in \{J, J+1, J+2, \ldots, N-1\}$ for which $k'_i < k_i$.*

Proof: [By Contraposition] We prove the contrapositive of the statement in Theorem 3.2.1: we assume that there does not exist $i \in \{J, J+1, \ldots, N-1\}$ such that $k'_i < k_i$ and show that, given this assumption, $W_q(K')$ has to be greater than or equal to $W_q(K)$.

Assume no $i \in \{J, J+1, \ldots, N-1\}$ exists for which $k'_i < k_i$. Then one of the following is true:

(a) $k_n = k'_n$ for all $n \in \{J, J+1, \ldots, N-1\}$, or

(b) there exists at least one $j \in \{J, J+1, \ldots, N-1\}$ such that $k'_j > k_j$, and the values of the rest of the switching points are the same in the two policies.

Case (a) implies that K' and K are the same policy, and so $W_q(K) = W_q(K')$.

To prove (b), suppose there exists exactly one $j \in \{J, J+1, J+2, \ldots, N-1\}$ such that $k'_j > k_j$, and $k_n = k'_n$ for all $n \in \{J, \ldots, N-1\} \setminus \{j\}$. Then K and K' are different in the value of exactly one switching point. Consequently, by Theorem 2.4.1,

$W_q(K') \geq W_q(K)$. Similarly, by applying Theorem 2.4.1 several times, the result generalizes to cases when there exists more than one j such that $k'_j > k_j$.

Therefore, if no $i \in \{J, J+1, \ldots, N-1\}$ exists for which $k'_i < k_i$, it follows that $W_q(K') \geq W_q(K)$. In other words, if $W_q(K') < W_q(K)$ then there exists $i \in \{J, J+1, \ldots, N-1\}$ for which $k'_i < k_i$. ■

This implies that, given a feasible policy $(0, 1, \ldots, k^*_i, k^*_{i+1}, \ldots, k^*_{N-1}, S)$ where all switching points with index i or greater are in the tail of the policy, we know that a solution with smaller W_q has to satisfy the constraint $((k_i < k^*_i) \;||\; (k_{i+1} < k^*_{i+1}) \;||\; \ldots \;||\; (k_{N-1} < k^*_{N-1}))$. Therefore, in order to implement the dominance rule, we add such a constraint during search every time a feasible policy is found, which should lead to a reduction in the size of the search space. Section 3.4 presents experimental results regarding the usefulness of this technique.

3.3 Shaving

Another inference method that is known to be useful in reducing the search space of a problem is shaving, which temporarily adds constraints to the problem, performs propagation and reduces variable domains based on the resulting state of the problem.

In a shaving procedure for our problem, we temporarily assign a particular value to a switching point variable, while the rest of the variables are assigned either their maximum or their minimum possible values. Depending on whether the resulting policies are feasible or infeasible, new bounds for the switching point variables may be derived. An example using the instance with $N = 3$, $S = 6$, $\lambda = 15$, $\mu = 3$, $B_l = 0.32$ is used below for illustration purposes. This is the same example as used by Berman et al. [15] to illustrate their heuristic. Policy $\hat{K}$, which always yields the smallest possible W_q [15], for this instance is $(k_0, k_1, k_2, k_3) = (0, 1, 2, 6)$ and policy $\hat{\hat{K}}$, which always yields the greatest possible W_q [15], is $(3, 4, 5, 6)$. Thus, the initial domains of the switching points are $[0..3]$, $[1..4]$, $[2..5]$ and $[6]$

for k_0, k_1, k_2 and k_3, respectively. At any step, shaving may be able to reduce the domains of one or more of these variables.

3.3.1 B_l-based Shaving Procedure

The initial shaving procedure consists of two cases in which either the upper or the lower bounds of variables may be modified.

In the first case, the constraint $k_i = \min(k_i)$, where $\min(k_i)$ is the smallest value from the domain of k_i, is temporarily added to the problem for some particular value of i between 0 and N. All other switching points are assigned their maximum possible values using the function *gMax*. Given an array of variables, the function *gMax* assigns the maximum possible values to all of the variables that do not yet have a value, returning true if the resulting assignment is feasible, and false otherwise. The maximum possible values are not necessarily the upper bound values in the domains of the corresponding variables, rather they are the highest values in these domains that respect the condition that $k_n < k_{n+1}$, $\forall n \in \{0, ..., N-1\}$. In the above example, if k_2 is assigned the value 3, while the rest of the variables are unbound, *gMax* would result in policy $(1, 2, 3, 6)$, which is infeasible, and thus false would be returned.

Recall that an assignment is infeasible when it yields a B value which is smaller than B_l. When the policy resulting from the addition of $k_i = \min(k_i)$ and the use of *gMax* is infeasible, and $\min(k_i) + 1 \leq \max(k_i)$, the constraint $k_i > \min(k_i)$ can be added to the problem: if all variables except k_i are set to their maximum values, and the problem is infeasible, then in any feasible policy k_i must be greater than $\min(k_i)$. Such reasoning is valid since Theorem 2.4.1 states that increasing the value of a switching point will increase B. Note that if the solution is feasible, it should be recorded as the best-so-far solution if its W_q value is smaller than the W_q value of the previous best policy. For easier reference, this part of the shaving procedure will be referred to as the *gMax* case.

In the second case, the constraint $k_i = \max(k_i)$ is added to the problem for some i between 0 and N, where $\max(k_i)$ is the maximum value from the domain of k_i. The rest of the variables

are assigned the minimum values from their domains using the function *gMin*. These assignments are made in a way that respects the constraints $k_n < k_{n+1}$, $\forall n \in \{0, ..., N-1\}$. If the resulting policy is feasible, then the constraint $k_i < \max(k_i)$ can be permanently added to the problem, assuming $\max(k_i) - 1 \geq \min(k_i)$. Since all variables except k_i are at their minimum values already, and k_i is at its maximum, it must be true, again by Theorem 2.4.1, that in any better solution the value of k_i has to be smaller than $\max(k_i)$. This case will be further referred to as the *gMin* case.

In both cases, if the inferred constraint violates the current upper or lower bound of a k_i, then the best policy found up to that point is optimal. On the contrary, whenever the domain of a switching point is modified as a result of inferences made during the *gMax* or *gMin* case, all of the switching points need to be re-considered by the shaving procedure. If the domain of one variable is reduced during a particular shaving iteration, some of the temporary constraints added in the next round of shaving will be different from the ones used previously, and, consequently, new inferences may be possible. Thus, the shaving procedure terminates when optimality is proven or when no more inferences may be made.

Consider now the example mentioned above. Suppose the constraint $k_0 = 0$ is added to the problem, and all the rest of the variables are assigned their maximum possible values (using *gMax*). The resulting policy is $(k_0, k_1, k_2, k_3) = (0, 4, 5, 6)$. This policy yields a B value of 0.63171, which implies that this policy is feasible, and so no domain reductions can be inferred. The constraint $k_0 = 0$ is then removed. Since the domain of k_0 has not been modified, the procedure considers the next variable. Thus, the constraint $k_1 = 1$ is added, and all the other variables are again set to their maximum values. The resulting policy is $(0, 1, 5, 6)$, which is also feasible since its B value is 0.508992. The constraint $k_1 = 1$ is then removed and $k_2 = 2$ is added. When all the other variables are set to their maximum values, the resulting policy is $(0, 1, 2, 6)$. This policy yields a B value of 0.1116577, which is smaller than 0.32. Thus, this policy is infeasible, and the constraint $k_2 > 2$ is added to the problem. This changes the domain of k_2 to $[3..5]$. Whenever the domain of a variable is reduced, the next shaving step

considers the same switching point, and so the next constraint to be added is $k_2 = 3$.

Now, consider the *gMin* case and assume that all variables have their full initial domains. Suppose the constraint $k_0 = 3$ is added to the problem. All the rest of the variables are assigned their smallest possible values consistent with $k_0 = 3$. Thus, the policy $(3, 4, 5, 6)$ is considered. This policy has a B value of 0.648305 and is feasible (it is in fact $\hat{\hat{K}}$, so if it were infeasible, the problem would have been infeasible). The value of k_0 in any better solution has to be smaller than 3, and so the domains of the variables become $[0..2]$, $[1..4]$, $[2..5]$, and $[6]$. The constraint $k_0 = 3$ is removed, and, since the domain of k_0 has been modified, the constraint $k_0 = 2$ is added next. The policy that is considered now is $(2, 3, 4, 6)$. This policy is also feasible, and so the domains become $[0..1]$, $[1..4]$, $[2..5]$, $[6]$. The temporary constraint $k_0 = 2$ is removed, and the next one added is $k_0 = 1$. The corresponding policy assigned by *gMin* is infeasible, and no domain reductions are made. Since the addition of $k_0 = 1$ did not result in any domain reductions, there is no need to reconsider the variable k_0 until after all other switching points have been looked at. Consequently, the next temporary constraint to be added is $k_1 = 4$.

In the complete B_l-based shaving procedure, we can start either with the *gMin* or the *gMax* case. Since policies considered in the *gMin* case will generally have smaller waiting time than ones considered in the *gMax* case, it may be beneficial to start with the *gMin* case. This is the approach we take.

Our complete B_l-based shaving algorithm is presented in Figure 3.1. It is assumed in all of the algorithms presented that the functions "add($constraint$)" and "remove($constraint$)" add and remove $constraint$ to and from the model, respectively.

Upon the completion of this shaving procedure, the constraint $W_q \leq bestW_q$, where $bestW_q$ is the value of the best solution found up to that point, is added ($W_q \leq bestW_q$ rather than $W_q < bestW_q$ is added because of issues with testing equality of floating point numbers). However, although such a constraint rules out policies with higher W_q as infeasible, it results in almost no propagation of the domains of the decision variables and does little to reduce the size of the search tree. In order to remedy this problem, another shaving procedure, this time

Algorithm 1: B_l***-based Shaving***
Input: S, N, μ, λ, B_l (problem instance parameters); $bestW_q$ (objective function value of the best solution found so far); $bestSolution$ (best solution found so far)
Output: $bestW_q$ and $bestSolution$ with (possibly) modified domains of the variables k_i, or $bestW_q$ and $bestSolution$ with proof of their optimality

$changed = true$
while ($changed$)
  $changed = false$
  for all i from 0 to $N-1$
    $successfulShave = true$
    while ($successfulShave$)
      $successfulShave = false$
      add($k_i = \max(Domain(k_i))$)
      if (*gMin*)
        if ($W_q \leq bestW_q$)
          $bestSolution = currentSolution$
          $bestW_q = W_q$
        if ($\max(Domain(k_i)) - 1 \geq \min(Domain(k_i))$)
          add($k_i < \max(Domain(k_i))$)
          $successfulShave = true$
          $changed = true$
        else
          return $bestSolution$, $bestW_q$; stop, optimality has been proven
      remove($k_i = \max(Domain(k_i))$)

    $successfulShave = true$
    while ($successfulShave$)
      $successfulShave = false$
      add($k_i = \min(Domain(k_i))$)
      if (*gMax*)
        if ($W_q \leq bestW_q$)
          $bestSolution = currentSolution$
          $bestW_q = W_q$
      else
        if ($\min(Domain(k_i)) + 1 \leq \max(Domain(k_i))$)
          add($k_i > \min(Domain(k_i))$)
          $successfulShave = true$
          $changed = true$
        else
          return $bestSolution$, $bestW_q$; stop, optimality has been proven
      remove($k_i = \min(Domain(k_i))$)

Figure 3.1: B_l-based shaving algorithm

based on the constraint $W_q \leq bestW_q$ is proposed in the next sub-section. The issue of the lack of propagation of the domains of k_i from the addition of this constraint will be discussed in more detail in Section 3.6.1.

3.3.2 W_q-based Shaving Procedure

The W_q-based shaving procedure makes inferences based strictly on the constraint $W_q \leq bestW_q$. In fact, the constraint $B \geq B_l$ is removed prior to running this procedure in order to eliminate the possibility of incorrect inferences. Similarly to the B_l-based shaving procedure, a constraint of the form $k_i = \max(k_i)$, where $\max(k_i)$ is the maximum value in the domain of k_i, is added temporarily, and the function *gMin* is used to assign values to the rest of the variables. As the B_l constraint has been removed, the only reason why the policy could be infeasible is because it has a W_q value greater than the best W_q that has been encountered so far. Since all switching points except k_i are assigned their smallest possible values, this implies that in any solution with a smaller expected waiting time, the value of k_i has to be strictly smaller than $\max(k_i)$. Note that this inference is correct regardless of whether the policy is feasible in terms of B_l or not. The complete procedure is stated in pseudo-code in Figure 3.2.

3.3.3 Combination of Shaving Procedures

W_q-based and B_l-based shaving will result in different domain reductions since they are based on two different constraints. Moreover, using the two together may cause more domain modifications than when either is used by itself. For example, when W_q-based shaving is successful, the upper bounds of some of the variables are reduced. As a result, if B_l-based shaving is used after such a run of W_q-based shaving, a completely new set of policies will be considered by the function *gMax*, possibly leading to new domain reductions. Similarly, if the B_l-based shaving procedure finds a new best solution, W_q-based shaving may be able to make new domain reductions due to the new value of $bestW_q$. Therefore, it makes sense to run the B_l-based and

Algorithm 2: W_q***-based Shaving***

Input: S, N, μ, λ, B_l (problem instance parameters); $bestW_q$ (objective function value of the best solution found so far); $bestSolution$ (best solution found so far)

Output: $bestW_q$ and $bestSolution$ with (possibly) modified domains of the variables k_i, or $bestW_q$ and $bestSolution$ with proof of their optimality

$changed = true$
while ($changed$)
  $changed = false$
  for all i from 0 to $N - 1$
    $successfulShave = true$
    while ($successfulShave$)
      $successfulShave = false$
      add($k_i = \max(Domain(k_i))$)
      if (!*gMin*)
        if ($\max(Domain(k_i)) - 1 \geq \min(Domain(k_i))$)
          add($k_i < \max(Domain(k_i))$)
          $successfulShave = true$
          $changed = true$
      remove($k_i = \max(Domain(k_i))$)

Figure 3.2: W_q-based shaving algorithm

W_q-based shaving procedures alternately (with W_q and B_l constraints added and removed appropriately) until no more domain pruning is possible. Such a combination of the two shaving procedures will be referred to as *AlternatingShaving*.

The *AlternatingShaving* procedure can be effectively combined with search in the following manner. *AlternatingShaving* can be run initially, until no further domain modifications are possible. Search can then be performed until a better solution is found, at which point *AlternatingShaving* can be applied again. Subsequently, search and shaving can alternate until one of them proves optimality of the best solution found. Such an approach may be successful because if search finds a new best solution, a new constraint on W_q will be added, and so W_q-based shaving may be able to reduce the upper bounds of the switching point variables. As stated previously, modifications of the upper bounds may then lead to further domain changes resulting from B_l-based shaving. This way of combining search and shaving will be further referred to as *AlternatingSearchAndShaving*.

One should note that other variations of shaving are possible. In particular, both B_l-based and W_q-based shaving procedures can be extended to make inferences about values of two switching points. For example, one can assign maximum values to a pair of switching point variables, while assigning minimum values to the rest. If the resulting policy is feasible, then a constraint stating that at least one variable from this pair has to be assigned a smaller value can be added to the problem. Various combinations of shaving procedures based on one switching point with ones based on two switching points are possible. However, preliminary experiments indicated that shaving procedures based on two switching points do not, in general, result in more effective models. This is because such procedures do not explicitly reduce the domains of the switching point variables but rather add a set of constraints to the model which do not appear to significantly reduce the search space. One possible direction for future work may be to further investigate these possible variations of shaving.

3.4 Experimental Results

Several sets of experiments were performed in order to evaluate the efficiency of the proposed models and the effectiveness of dominance rules and shaving procedures, as well as to compare the performance of the best CP model with the performance of heuristic P_1 proposed by Berman et al. [15] (see Section 2.4.1.2 for a description of this heuristic). All constraint programming models were implemented in ILOG Solver 6.2,[5] while the heuristic of Berman et al. was implemented using C++.

Recall that policy $\hat{K} = \{k_0 = 0, k_1 = 1, k_2 = 2, \ldots, k_{N-1} = N-1, k_N = S\}$ yields the smallest possible W_q and B, and the policy $\hat{\hat{K}} = \{k_0 = S-N, k_1 = S-N+1, \ldots, k_{N-1} = S-1, k_N = S\}$ yields the greatest possible W_q and B. The information gained from these two policies is explicitly used in the implementation of all three models. In particular, the feasibility of $\hat{\hat{K}}$ is checked first. If this policy is infeasible, then the program stops as there is no feasible solution for that instance [15]. If $\hat{\hat{K}}$ is feasible, the feasibility of $\hat{K}$ is checked next. If $\hat{K}$ is feasible, then it is optimal [15]. The two cases when $\hat{K}$ is optimal and $\hat{\hat{K}}$ is infeasible are therefore trivial and are solved easily both by the CP models and by Berman et al.'s heuristic P_1. Additionally, experiments with instances in which $\hat{\hat{K}}$ is optimal are not presented. Although these are very hard to solve without shaving, using the elementary B_l-based shaving procedure will always result in a (usually fast) proof of the optimality of this policy. This case is also trivial for Berman et al.'s heuristic. Consequently, the experimental results presented here are based only on the instances for which the optimal is between $\hat{K}$ and $\hat{\hat{K}}$.

Preliminary experiments indicated that the value of S has a significant impact on the efficiency of the programs since higher values of S result in larger domains for the k_i variables for all models and a higher number of w_j variables for the *Dual* model. As indicated in Table 3.1 in Section 3.1.5, S also has a big impact on the number of constraints in the *If-Then* and

[5] Results are quite sensitive to the level of precision that is set. In all models, we set the default Solver precision to 0.000001.

Dual models. Therefore, it was essential to consider instances for each value of S from the set $\{10, 20, \ldots, 100\}$ in order to gain an accurate understanding of the performance of the model and the heuristic. Thirty instances were generated for each S in such a way as to ensure that the instance is feasible and that the optimal policy is neither $\hat{\hat{K}}$ nor $\hat{K}$.[6] A 10-minute time limit on the overall run-time of the program was enforced in the experiments. All experiments were performed on a Dual Core AMD 270 CPU with 1 MB cache, 4 GB of main memory, running Red Hat Enterprise Linux 4.

In order to perform comparisons between the CP models, and between the CP models and Berman et al.'s heuristic, we look at mean run-times, the number of instances in which the optimal solution was found, the number of instances in which optimality was proven and the mean relative error (MRE). MRE is a measure of solution quality that allows one to observe how quickly a particular algorithm is able to find a good solution. MRE is defined as

$$MRE(a, M) = \frac{1}{\mid M \mid} \sum_{m \in M} \frac{c(a, m) - c^*(m)}{c^*(m)} \tag{3.18}$$

where a is a particular algorithm that is used to solve the problem, M is the set of problem instances on which the algorithm is being tested, $c(a, m)$ is the cost of a solution found for instance m by algorithm a, and $c^*(m)$ is the best solution for instance m found during our experiments.

3.4.1 Comparison of Constraint Programming Models and Techniques

Each CP model was tested with and without the use of shaving and dominance rules. A total of 30 CP-based methods for solving this problem were therefore evaluated. In the following results, a model with B_l-based shaving is a model which runs the B_l-based shaving procedure until no more domain changes are possible, adds a constraint on the value of W_q based on the best solution found during the shaving procedure and runs search for the rest of the time. Similarly,

[6] For most instances with S greater than 100, neither our method nor Berman et al.'s heuristic P_1 may be used due to numerical instability. The maximum value of S used in the experiments of Berman et al. is also 100.

models with W_q-based shaving and *AlternatingShaving* are models which run the W_q-based shaving procedure and the *AlternatingShaving* procedure, respectively, until it is no longer possible to reduce the domains of the switching point variables, add a constraint requiring W_q to be less than the expected waiting time of the best solution found by the shaving procedure and use search for the rest of the time. As described previously, *AlternatingSearchAndShaving* alternates between search and the *AlternatingShaving* procedure. In all models, search assigns switching points in increasing index order. The smallest value in the domain of each variable is tried first. In future work, it may be worthwhile to experiment with other variable and value ordering heuristics to see if the performance of the models can be improved.

3.4.1.1 Constraint Programming Models

Table 3.2 presents the number of instances, out of 300, for which the optimal solution has been found and proven by each of the 30 proposed CP-based methods. This table indicates that the *PSums* model is the most effective of the three, proving optimality in the largest number of instances regardless of the use of dominance rules and shaving. With *AlternatingSearchAnd-Shaving*, *PSums* proves optimality in the largest number of instances: in 79.3% of all instances, or 238 out of 239 instances for which optimality has been proven by any model.

Observations from Table 3.2 can be further confirmed by looking at Figure 3.3. This figure shows how MRE changes over the first 50 seconds of run-time for *If-Then*, *PSums* and *Dual* models with *AlternatingSearchAndShaving*, and for Berman's heuristic (we comment on the performance of the heuristic in Section 3.4.2). It can be seen that *PSums* is, on average, able to find better solutions than the other two models given the same amount of run-time.

In Table 3.3, additional statistics regarding the performance of the three models with *AlternatingSearchAndShaving* and without dominance rules are presented (we comment on the same statistics for the heuristic in Section 3.4.2). In particular, for each model, the number of instances in which it finds the best solution (out of 300), the number of instances in which it finds the optimal solution (out of all cases for which optimality has been proven) and the

	No Shaving		B_l-based Shaving		W_q-based Shaving		*Alternating Shaving*		*AlternatingSearch AndShaving*	
	D	ND	D	ND	D	ND	D	ND	D	ND
If-Then	105	105	192	191	105	105	219	218	234	234
PSums	126	126	202	201	126	126	225	225	238	238
Dual	105	105	191	191	105	105	218	218	232	232

Table 3.2: Number of instances for which optimality is found and proven within 10 CPU-minutes out of a total of 300 problem instances (D - with dominance rules, ND - without dominance rules).

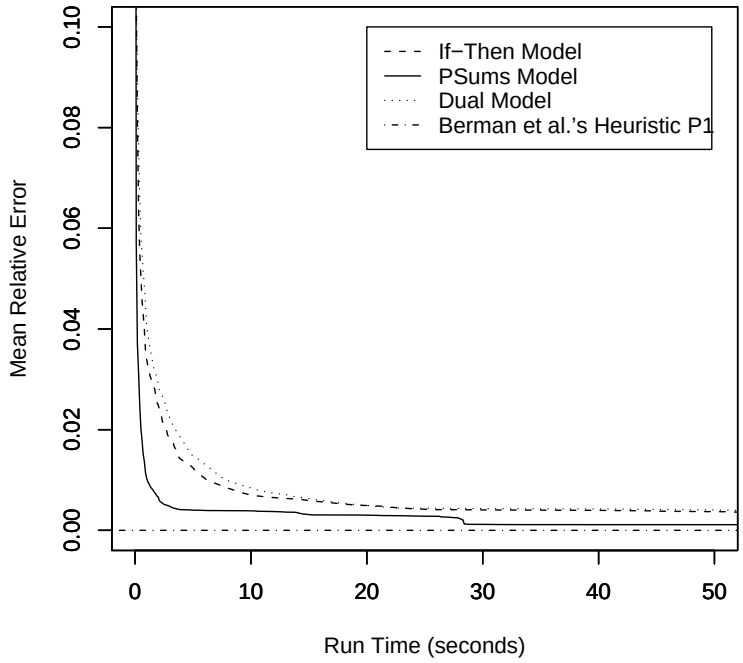

Figure 3.3: Comparison of MRE of three CP models with *AlternatingSearchAndShaving* with Berman's Heuristic P_1.

number of times it proves optimality are presented. It can be seen that all models find the optimal solution in the 239 instances for which it is known. However, the *PSums* model proves optimality in 4 more instances than the *If-Then* model and in 6 more instances than the *Dual*. *PSums* also finds the best-known solution of any algorithm in 97.6% of all the instances considered. A more detailed discussion of the differences in the performance of the CP models is presented in Section 3.6.2.

3.4.1.2 Shaving Procedures

From Table 3.2, it can be observed that without shaving and with the W_q-based shaving procedure, the CP models prove optimality in the fewest number of cases. The similarity in performance of models without shaving and with W_q-based shaving is not surprising because the W_q-based procedure is able to start pruning domains only when the value of the best policy found up to that point is quite good. When the W_q-based procedure is used alone, it only has one solution to base its inferences on, namely $\hat{\hat{K}}$. Since all policies will result in a smaller expected waiting time than $\hat{\hat{K}}$, this procedure by itself is useless.

Employing the B_l-based shaving procedure substantially improves the performance of all models: without dominance rules, the *If-Then*, the *PSums* and the *Dual* models prove optimality in 86, 75 and 86 more instances, respectively, than the corresponding models without

	# best found (/300)	# optimal found (/239)	# optimal proved (/300)
PSums	293	239	238
If-Then	282	239	234
Dual	279	239	232
P_1	282	238	0

Table 3.3: Comparison of three CP models (with *AlternatingSearchAndShaving* and without dominance rules) with Berman's Heuristic P_1.

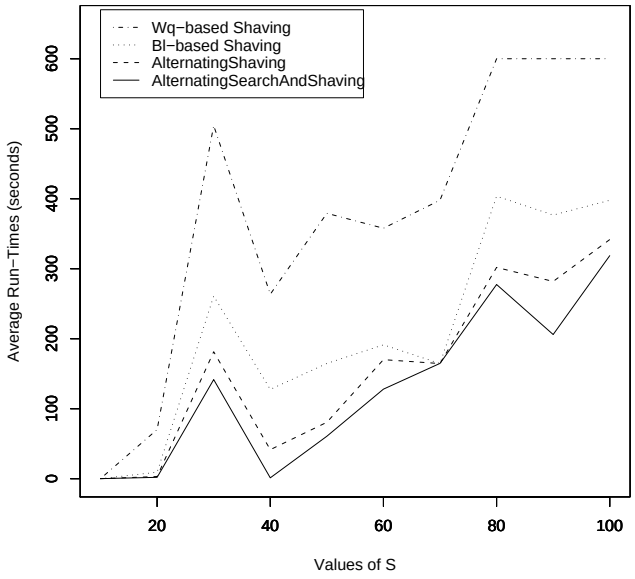

Figure 3.4: *PSums* model with various shaving techniques: average run-times for each value of S. Average run-times for *PSums* without shaving are not shown in this graph since the resulting curve would be indistinguishable from the one for W_q-based shaving.

shaving or with only W_q-based shaving; with dominance rules, the situation is equivalent. These results imply that inferences made based on the B_l constraint are effective in reducing the domains of the decision variables.

Models with *AlternatingShaving* and *AlternatingSearchAndShaving* perform even better than models employing the B_l-based shaving procedure. This leads to the observation that the real power of W_q-based shaving only becomes apparent when it is combined with B_l-based shaving. This is because B_l-based shaving often finds a good solution and a good value of $bestW_q$, allowing the W_q-based shaving procedure to infer more domain reductions. Similarly, this observation explains why *AlternatingSearchAndShaving* performs even better than *AlternatingShaving*. In particular, in *AlternatingSearchAndShaving*, the W_q-based procedure is used both after a new best solution is found during shaving and after one is found during

search. Therefore, if a higher quality solution is found during search, it will be used by the W_q-based procedure to further prune the domains of the switching point variables.

In Figure 3.4, the average run-times for each value of S from 10 to 100 are presented for the four shaving procedures with the *PSums* model.[7] This graph shows that, for each value of S, the *AlternatingSearchAndShaving* procedure gives the best performance. Additionally, this graph shows that as S increases, it becomes increasingly difficult to prove optimality and average run-time increases. As stated previously, this is due to larger domains of the switching point variables. The *AlternatingSearchAndShaving* procedure, however, is able to significantly reduce the domains of the k_i variables and therefore provides an effective method for instances with higher values of S as well.

3.4.1.3 Dominance Rules

Table 3.2 indicates that there is rarely any difference in the number of instances solved to optimality between a model with dominance rules and a model without dominance rules. No difference at all is visible for any model without shaving, with W_q-based shaving and *AlternatingSearchAndShaving*. Recall that dominance rules are implemented by the addition of a constraint on the values of the switching point variables after a solution is found. Such a constraint will be more effective when more of the switching point variables are assigned their minimum values in the current solution. Usually, such policies are also the ones which result in a smaller expected waiting time. Similarly, W_q-based shaving is useful only when a solution with small expected waiting time is found. This leads to the conjecture that dominance rules may be effective only for the same instances for which the W_q-based shaving procedure is effective. This conjecture is supported by the results of Table 3.2. In particular, when the W_q-based shaving procedure is used by itself, it makes inferences based only on the policy $\hat{\hat{K}}$, a solution that is generally of poorest quality for an instance. The method with a single run of

[7] Since a run-time limit of 600 seconds was used throughout the experiments, we assumed a run-time of 600 seconds for all instances for which optimality has not been proven within this limit. Therefore, the mean run-times reported throughout this chapter are underestimates of the true means.

W_q-based shaving therefore heavily relies on search. Since search takes a long time to find a feasible solution of good quality, the effectiveness of dominance rule-based constraints is also not visible within the given time limit.

On the other hand, in *AlternatingSearchAndShaving*, the W_q-based procedure plays a key role because it makes domain reductions based on high quality solutions produced by B_l-based shaving, and later, search. Dominance rules do not play a role in this procedure since shaving is used after every new solution found by search. However, even if dominance rule constraints were explicitly incorporated in the procedure (i.e. if they were added before each new run of search), they would be redundant since they serve essentially the same purpose as the W_q-based shaving procedure.

When shaving is not used, the results are equivalent to those achieved when W_q-based shaving is employed. The explanation for the absence of a difference between models with and without dominance rules is therefore also the same. In particular, it takes a long time for a solution to be found whose quality is such that it allows the dominance rule constraint to effectively reduce the size of the search tree. Given a longer time limit, it is possible that a difference may have become visible.

When B_l-based shaving and *AlternatingShaving* are used, dominance rules are sometimes helpful. In both cases, this is because after these two shaving procedures, subsequent search usually finds a good solution quickly, and, since W_q-based shaving is not used again at that point, the dominance rule constraint that is added can be effective in reducing the size of the search tree.

Overall, it can be observed that using *AlternatingSearchAndShaving* without dominance rules is more effective than using B_l-based shaving or *AlternatingShaving* with dominance rules. Therefore, in further comparisons, the focus is only on models with *AlternatingSearchAndShaving* without dominance rules.

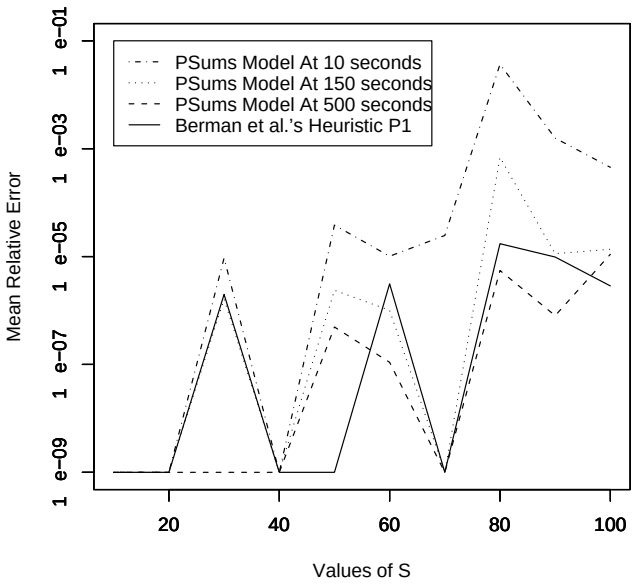

Figure 3.5: MRE for each value of S for P_1 and the $PSums$ model.

3.4.2 Heuristic P_1 vs. Best Constraint Programming Approach

Empirical results regarding the performance of heuristic P_1 are not presented in the paper of Berman et al. [15], and so the ability of P_1 to find good switching policies has not been explicitly evaluated in previous work. We wanted to find out how well the heuristic actually performs by comparing it to our CP methods, and in particular, to our best CP method–*PSums* model with *AlternatingSearchAndShaving* and without dominance rules.

In Table 3.3, we present several measures of performance for the three proposed models with *AlternatingSearchAndShaving* and for the heuristic P_1. It can be seen that the heuristic performs well, finding the best-known solution in only eleven fewer instances than the *PSums* model, in three more instances than the *Dual* model and in the same number of instances as the *If-Then* model. Moreover, the heuristic finds, but, of course, cannot prove, the optimal solution in 238 out of 239 instances for which the optimal is known. The three CP models find the optimal solution in all 239 of these. It should also be noted that the run-time of the heuristic is negligible (close to 0 seconds), whereas the mean run-time of the *PSums* model is approximately 130 seconds (the mean run-times of the other two models are slightly higher: 141 seconds for the *If-Then* model and 149 seconds for the *Dual* model).

Table 3.3 also shows that the *PSums* model is able to find the best-known solution in 11 more instances than the heuristic. Closer examination reveals that there are 275 instances in which both the *PSums* model and P_1 find the best-known solution, 18 instances in which only *PSums* is able to do so and 7 in which only the heuristic finds the optimal.

From Figure 3.3, it can be observed that the heuristic achieves a very small MRE in a negligible amount of time. After about 50 seconds of run-time, the MRE over 300 instances resulting from *PSums* with *AlternatingSearchAndShaving* becomes comparable to that of the heuristic MRE. In Figure 3.5, the MRE over 30 instances for each value of S is presented for the heuristic and for *PSums* with *AlternatingSearchAndShaving* at 10, 150 and 500 seconds of run-time. After 10 seconds, the performance of *PSums* is comparable to that of the heuristic for values of S smaller than or equal to 40, but the heuristic appears to be quite a bit better for

higher values of S. At 150 seconds, the performance of *PSums* is comparable to that of the heuristic except at S values of 50 and 80. After 500 seconds, *PSums* has a smaller MRE over the 300 instances and also a lower (or equal) MRE for each value of S except 50 and 100.

Overall, these results indicate that the heuristic performs well–its run-time is negligible, it finds the optimal solution in all but one of the cases for which it is known, and it finds the best solution in 94% of all instances. Moreover, it results in very low MRE. Although *PSums* with *AlternatingSearchAndShaving* is able to achieve slightly higher numbers in most of the performance measures, it is clear that these improvements are small given that the *PSums* run-time is so much higher than the run-time of the heuristic.

3.5 *PSums*-P_1 Hybrid

Naturally, it is desirable to create a method that would be able to find a solution of high quality in a very short amount of time, as does Berman's heuristic, and that would also have the same high rate of being able to prove optimality within a reasonable run-time as does *PSums* with *AlternatingSearchAndShaving*. It is therefore worthwhile to experiment with a *PSums*-P_1 Hybrid, which starts off by running P_1 and then, assuming the instance is feasible, uses the *PSums* model with *AlternatingSearchAndShaving* to find a better solution or prove the optimality of the solution found by P_1 (infeasibility of an instance is proven if the heuristic determines that policy $\hat{\hat{K}}$ is infeasible).

Since it was shown that heuristic P_1 is very fast, running it first incurs almost no overhead. Throughout the analysis of experimental results, it was also noted that the performance of the W_q-based shaving procedure depends on the quality of the best solution found before it is used. We have shown that the heuristic provides solutions of very high quality (in 94% of instances used in the experiments, it found the best solution). Therefore, the first iteration of the W_q-based procedure should be able to significantly prune the domains of switching point variables because of the good-quality solution found by the heuristic. Continuing by alternating the two

shaving techniques and search, which has also been shown to be an effective approach, should result in at least as many instances for which optimality is proven as for the *PSums* model with *AlternatingSearchAndShaving*.

The proposed Hybrid algorithm was tested on the same set of 300 instances that was used above. Results illustrating the performance of the Hybrid as well as the performance of P_1 and *PSums* with *AlternatingSearchAndShaving* are presented in Table 3.4. The Hybrid was able to find the best solution in all 300 cases while finding the optimal solution and proving optimality in as many instances as the *PSums* model. Thus, in spite of the good-quality solutions that are discovered quickly because of the heuristic, the domains of switching points in some instances are not reduced significantly enough to increase the number of cases in which optimality is proven. The mean run-time for the Hybrid is essentially identical to the mean run-time of *PSums* with *AlternatingSearchAndShaving*, equalling approximately 130 seconds.

Thus, the Hybrid is the best choice for solving this problem: it finds as good a solution as the heuristic in as little time (close to 0 seconds), it is able to prove optimality in as many instances as the best constraint programming method, and it finds the best-known solution in all instances considered. Moreover, all these improvements are achieved without an increase in the average run-time over the *PSums* model.

	# best found (/300)	# optimal found (/239)	# optimal proved (/300)
PSums	293	239	238
P_1	282	238	0
PSums-P_1 Hybrid	300	239	238

Table 3.4: Comparison of *PSums* model with *AlternatingSearchAndShaving* and Berman et al.'s Heuristic P_1 with the Hybrid model.

3.6 Discussion

In this section, we examine some of the reasons for the poor performance of the CP models without shaving and suggest reasons for the observed differences among the CP models.

3.6.1 Lack of Back-Propagation

In our experiments, we have some instances for which even the *PSums*-P_1 Hybrid with *AlternatingSearchAndShaving* is unable to prove the optimality of the best solution found within the 10-minute time limit. In fact, in many of these instances, the amount of time spent during search is higher than the time spent on shaving, and the run-time limit is usually reached during the search, rather than the shaving, phase. Further analysis of the algorithms' behaviour suggests that this poor performance of search can be explained by the lack of back-propagation. Back-propagation refers to the pruning of the domains of the decision variables due to the addition of a constraint on the objective function: the objective constraint propagates "back" to the decision variables, removing domain values and so reducing search. In the CP models presented above, there is very little back-propagation.

Consider a model without shaving. Throughout search, if a new best solution is found, the constraint $W_q \leq bestW_q$, where $bestW_q$ is the new objective value, is added to the model. However, the domains of the switching point variables are usually not reduced in any way after the addition of such a constraint. This can be illustrated by observing the amount of propagation that occurs in the *PSums* model when W_q is constrained.

For example, consider an instance of the problem with $S = 6$, $N = 3$, $\lambda = 15$, $\mu = 3$, and $B_l = 0.32$ (this instance is used in Section 3.3 to illustrate the shaving procedures). The initial domains of the switching point variables are $[0..3]$, $[1..4]$, $[2..5]$ and $[6]$. The initial domains of the probability variables $P(k_i)$ for each i, after the addition of W_q bounds provided by $\hat{K}$ and $\hat{\hat{K}}$, are listed in Table 3.5. The initial domain of W_q, also determined by the objective function values of $\hat{K}$ and $\hat{\hat{K}}$, is $[0.22225..0.425225]$. The initial domains of L and F, are

$[2.8175e^{-7}..6]$ and $[0..2.68]$, respectively. Upon the addition of the constraint $W_q \leq 0.306323$, where 0.306323 is the known optimal value for this instance, the domain of W_q is reduced to $[0.22225..0.306323]$, the domain of L becomes $[1.68024..6]$ and the domain of F remains $[0..2.68]$. The domains of $P(k_i)$ after this addition are listed in Table 3.5. The domains of both types of probability variables are reduced by the addition of the new W_q constraint. However, the domains of the switching point variables remain unchanged. Therefore, even though all policies with value of W_q less than 0.306323 are infeasible, constraining W_q to be less than or equal to this value does not result in any reduction of the search space. It is still necessary to search through all policies in order to show that no better feasible solution exists.

One of the reasons for the lack of pruning of the domains of the k_i variables due to the W_q constraint is likely the complexity of the expression for W_q. In particular, recall that W_q is expressed in all models as $W_q = \frac{L}{\lambda(1-P(k_N))} - \frac{1}{\mu}$. In the example above, when W_q is constrained to be less than or equal to 0.306323, we get the constraint $0.306323 \geq \frac{L}{15(1-P(k_N))} - \frac{1}{3}$, which implies that $9.594845(1 - P(k_N)) \geq L$. This explains why the domains of both L and $P(k_N)$ change upon this addition to the model. The domains of the rest of the $P(k_i)$ variables change because of the relationships between $P(k_i)$s (Equation (3.6)) and because of the constraint that the sum of all probability variables has to be 1. Similarly, the domains of $PSums(k_i)$ change because these variables are expressed in terms of $P(k_i)$ (Equation (3.8)). However, because the actual k_i variables mostly occur as exponents in expressions for $PSums(k_i)$, $P(k_i)$, and $L(k_i)$, the minor changes in the domains of $PSums(k_i)$, $P(k_i)$, or $L(k_i)$ that happen due to the constraint on W_q have no effect on the domains of the k_i. This analysis suggests that it may be interesting to investigate a CP model based on log-probabilities rather than on the probabilities themselves. Such a model may lead to stronger propagation.

Likewise, in the *If-Then* and *Dual* models, the domains of the decision variables are not reduced when a bound on the objective function value is added, although the domains of all probabilities, L and F are modified. In both of these models, the constraints relating F, L and the probability variables to the variables k_i are the balance equations, which are quite complex.

	Before addition of $W_q \leq 0.306323$		After addition of $W_q \leq 0.306323$	
j	$P(j)$	$PSums(j)$	$P(j)$	$PSums(j)$
k_0	$[4.40235e^{-6}..0.979592]$	$[0..1]$	$[4.40235e^{-6}..0.979592]$	$[0..0.683666]$
k_1	$[1.76094e^{-7}..1]$	$[0..1]$	$[0.000929106..1]$	$[0..0.683666]$
k_2	$[2.8175e^{-8}..0.6]$	$[2.8175e^{-8}..1]$	$[0.0362932..0.578224]$	$[0.0362932..0.71996]$
k_3	$[4.6958e^{-8}..1]$	N/A	$[0.28004..0.963707]$	N/A

Table 3.5: Domains of $P(j)$ and $PSums(j)$ variables for $j = k_0, k_1, k_2, k_3$, before and after the addition of the constraint $W_q \leq 0.306323$.

The domains of probability variables do not seem to be reduced significantly enough due to the new W_q bound so as to result in the pruning of k_i domains because of these constraints.

These observations served as the motivation for the proposed shaving techniques. In particular, the W_q-based shaving procedure reduces the domains of switching point variables when it can be shown that these values will necessarily result in a higher W_q value than the best one found up to that point. This makes up for some of the lack of back-propagation. However, even when this procedure is used after each new best solution is found, as in *AlternatingSearchAnd-Shaving*, it is not always able to prune enough values from the domains of the k_is so as to be able to prove optimality within 10 minutes of run-time. It can therefore be seen that inferences based on the value of W_q are very limited in their power and, therefore, if the domains of switching point variables are large after shaving, then it will not be possible to prove optimality in a short period of time.

3.6.2 Differences in the Constraint Programming Models

Experimental results demonstrate that the *PSums* model is the best out of the three models both with and without shaving. In order to understand the reasons behind such differences in performance, we examine the mean number of choice points statistics for the models without

	21 Instances Solved by *PSums* Only		105 Instances Solved by All Models	
	First Solution	Total	First Solution	Total
If-Then	8234	137592	1201	37596
PSums	6415	464928	1064	36590
Dual	7528	102408	1132	36842

Table 3.6: Mean number of choice points explored before the first solution is found and mean total number of choice points explored within 600 seconds for the three models without shaving and without dominance rules. For *PSums*, the latter statistic corresponds to the total number of choice points needed to prove optimality.

shaving. The mean number of choice points, or, equivalently, the mean number of nodes in the search tree, gives an indication of the size of the search space. In order to compare our three models, we look at the mean number of choice points considered before the first feasible solution is found and the mean total number of choice points explored within 600 seconds of run-time.

In Table 3.2, it is shown that, without shaving, the *PSums* model proves optimality in 21 more instances than the other two models and that there are 105 instances in which all three models prove optimality. In Table 3.6, we present the mean number of choice points statistics for all three models for both of these sets of instances. It can be seen that the mean number of choice points that need to be explored by the *PSums* model in order to find an initial solution is smaller than for the other two models, both for the 105 instances that are eventually solved to optimality by all models and for the 21 instances that are only solved to optimality by *PSums*. Because the same variable and value ordering heuristics are used in all models, this observation seems to imply that more propagation occurs during search in the *PSums* model than in the other two models. This claim is further supported by the fact that the mean total number of choice points for the 105 instances that are solved by all models is smaller for *PSums* than for the other two models.

Table 3.6 also shows that, for the 21 instances that are only solved to optimality by *PSums*, the mean total number of choice points is the highest for the *PSums* model. Since *PSums* is the only one out of the three models to solve these instances, this implies that propagation is happening faster in this model. In fact, this observation is confirmed by the results from the 105 instances that are solved by all three models: for these instances, the *Dual* explores an average of 713 choice points per second, the *If-Then* model explores an average of 895 choice points per second and the *PSums* model explores an average of 1989 choice points per second. In other words, it appears that propagation in the *PSums* model is more than twice as fast as in the other two models.

Overall, this examination indicates that the superiority of the *PSums* model without shaving is caused by its ability to remove more values (stronger propagation) and, more importantly, by the fact that propagation occurs faster in this model.

When shaving is employed, the *PSums* model also performs better than the *Dual* and the *If-Then* models, proving optimality in a greater number of instances (see Table 3.2) and also finding good-quality solutions faster (see Figure 3.3). In all models, the shaving procedures make the same number of domain reductions because shaving is based on the W_q and B_l constraints, which are present in all models. However, the time that each shaving iteration takes is different in different models. Our empirical results show that each iteration of shaving takes a smaller amount of time with the *PSums* model than with the other two models. Thus, with shaving, the *PSums* model performs better than the other two both because shaving is faster and because subsequent search, if it is necessary, is faster.

PSums is radically different from the other two models because it does not include an explicit representation of the balance equations and has no if-then constraints. Moreover, *PSums* has a smaller number of probability variables than the other two models ($2N + 1$ compared to $S + 1$ in the other two models – see Table 3.1), because it calculates sums of probabilities between two switching points rather than the probability of j customers being present in the front room for all j from 0 to S. In addition, the probability variables included in this model

are more tightly linked by the closed-form expressions. Thus, faster propagation in *PSums* appears to be the result of the tighter links between variables, the small number of probability variables, and the lack of if-then constraints.

A comparison of the *If-Then* model with *AlternatingSearchAndShaving* with the *Dual* with *AlternatingSearchAndShaving* using Figure 3.3 shows that the *If-Then* model is usually able to find good solutions in a smaller amount of time. Moreover, as shown in Table 3.3, the *If-Then* model with *AlternatingSearchAndShaving* finds the best solution in three more instances, and proves optimality in two more instances, than the *Dual* model with the same shaving procedure. With other shaving procedures and without shaving, the same statistics show almost no difference in the performance of the two models. It was expected that the *Dual* would outperform the *If-Then* model because it uses a simpler representation of the balance equations and expressions for F and L, and has a smaller number of if-then constraints (there are $S + 1$ such constraints in the *Dual*, compared to $3NS + 2N + S$ in the *If-Then* model). Table 3.6 shows that the *Dual* has to explore a smaller number of choice points to find the initial solution. In the 105 instances that both of these models solve, the total number of choice points explored by the *Dual* is also smaller, indicating that the search space is usually smaller in the *Dual* model. However, the *If-Then* model is faster, exploring, on average, 895 choice points per second compared to the average of 713 choice points per second explored by the *Dual*. One possible explanation for the *Dual* being slower is the fact that it has to assign more variables (via propagation) than the other models. In particular, in order to represent a switching policy, the *Dual* has to assign $S + 1$ w_j variables in addition to $N + 1$ k_i variables, with S usually being much larger than N.

Overall, it can be said that the *If-Then* and the *Dual* models are comparable in their performance and that *PSums* is a more effective choice than either of them regardless of whether shaving is employed or not.

3.7 Conclusions

In this chapter, a constraint programming approach is proposed for the problem of finding the optimal times to switch workers between the front room and the back room of a retail facility under stochastic customer arrival and service times. This is the first work of which we are aware that examines solving such stochastic queueing problems using constraint programming. The best pure CP method proposed is able to prove optimality in a large proportion of instances within a 10-minute time limit. Previously, there existed no non-heuristic solution to this problem. As a result of our experiments, we hybridized the best pure CP model with the heuristic proposed for this problem in the literature. This hybrid technique is able to achieve performance that is equivalent to, or better than, that of each of the individual approaches alone: it is able to find very good solutions in a negligible amount of time due to the use of the existing heuristic and to prove optimality in a large proportion of the problem instances within 10 CPU minutes due to the CP model.

This chapter demonstrates that constraint programming can be a good approach for solving a queueing control optimization problem. This illustrates that for queueing problems for which optimality is important or heuristics do not perform well, CP may prove to be an effective methodology.

Chapter 4

Finding the Optimal Staff Mix in a Retail Facility

In the previous chapter, it was shown that constraint programming can be used to solve a queueing control problem. The main purpose of this chapter is to show that constraint programming can also be used to solve a problem which deals with both queue design and queue control.

Similarly to Chapter 3, we consider a retail facility with back room and front room operations. We suppose that the retail facility has the option of hiring cross-trained workers, who are able to perform tasks in both rooms, or hiring specialized workers, who are able only to serve customers or only to perform back room work. Cross-trained workers give the facility more flexibility since they may be switched between the back room and the front room depending on the amount of work in the back room and the number of customers in the front room. However, cross-trained workers are usually more expensive than specialized ones because they possess more skills.

Managers of the facility are interested in finding the number of cross-trained and specialized workers that have to be hired so as to minimize the total staffing costs while at the same time ensuring that the expected waiting time of customers in the front room is bounded from above and that the expected number of workers in the back room is sufficient to complete all

required work. If cross-trained workers are employed, managers also need to know how to switch these workers between the two rooms in order to satisfy the back room and front room constraints.

In this chapter, the problem is solved using a constraint programming (CP) method based on Benders' decomposition.[1] The chapter demonstrates the applicability of constraint programming for solving queueing design and control problems and presents a complete method for an extension of Berman et al.'s problem P_2 [15] (P_2 is discussed in Sections 2.4.2 and 4.1.1).

4.1 Problem Description

As in the previous chapter, let S denote the maximum number of customers allowed in the front room at any one time. This implies that when there are S customers present, arriving customers will be blocked from joining the front room queue. The minimum expected number of workers that is required to be present in the back room in order to complete all of the work is assumed to be known and is denoted by B_l, where l stands for 'lower bound'. W_u denotes the upper bound on the expected customer waiting time, W_q. It is assumed that only one worker is allowed to be switched at a time, and both switching time and switching cost are negligible. Customers arrive according to a Poisson process with rate λ. Service times by workers in the front room follow an exponential distribution with rate μ.

4.1.1 Berman et al.'s Problem P_2

In Berman et al.'s problem P_2 (see Section 2.4.2 for additional details), the goal is to find the minimum number of cross-trained workers, N, required in the retail facility, subject to constraints which ensure that the expected customer waiting time is bounded from above and that there are enough workers in the back room in order to complete all of the back room work.

A switching policy, which would tell the managers how to switch workers between the two

[1] A shorter version of this chapter will appear as [94].

rooms, is defined in Berman et al.'s work [15] in terms of quantities k_i, for $i = 0, \ldots, N$. A policy $(k_0, k_1, \ldots, k_N)$ states that there should be i workers in the front room whenever there are between $k_{i-1} + 1$ and k_i customers in the front room, for $i = 1, 2, \ldots, N$. Given a family of switching policies, $\mathbf{K} = \{K; K = \{k_0, k_1, ..., k_{N-1}, S\}$, k_i integers, $k_i - k_{i-1} \geq 1$, $k_0 \geq 0, k_{N-1} < S\}$, problem P_2 is formally stated in Berman's paper [15] as

$$\begin{aligned} &minimize_{K \in \mathbf{K}}\ N \\ &s.t.\ W_q \leq W_u \\ &B \geq B_l. \end{aligned} \tag{4.1}$$

4.1.2 An Extension of P_2

In a direct extension of problem P_2, we suppose that the facility is able to hire specialized front room workers and/or specialized back room workers in addition to the cross-trained ones. Because of the skill differences, each type of worker may have a different staffing cost. The goal of this problem is therefore to find the lowest-cost combination of specialized and cross-trained workers so as to ensure that the expected waiting time of customers does not exceed W_u and that there are enough workers in the back room to ensure that all back room work is completed.

Let f be the number of specialized front room workers, b the number of specialized back room workers, and x the number of cross-trained workers in the facility. It is assumed that the service rate of a worker in the front room is equal to μ regardless of whether this worker is specialized or cross-trained. Similarly, cross-trained and specialized employees working in the back room are assumed to be able to perform back room tasks equally well.

Denote the staffing costs as c_f, c_b and c_x, respectively for front room, back room and cross-trained workers. We assume that $c_x \geq c_f > 0$, $c_x \geq c_b > 0$ and $c_x \leq c_f + c_b$. Such assumptions are reasonable given that in many situations a cross-trained worker is more expensive than a

specialist. Moreover, under other cost assumptions, the problem may be easier to solve. For example, if we assume that $c_x \geq c_f + c_b$, it can be noted that there should be no cross-trained workers in the optimal staffing configuration, since hiring a pair of specialized workers instead of one cross-trained worker always leads to equivalent or smaller staffing cost. A detailed discussion of the problem under different cost assumptions can be found in Chapter 5.

Given particular values of x, f and b, and a policy specifying when cross-trained workers are to be switched between the two rooms, one can calculate the expected customer waiting time, W_q, and the expected number of workers in the back room, B. (See below for the definition of a switching policy and expressions for the calculation of B and W_q). Since we are required to find both the optimal staff mix and a satisfying switching policy, this problem is both a queueing design and a queueing control problem.

The problem of finding the optimal staff mix given front room and back room constraints can be stated as

$$\begin{aligned} &minimize\ c_f f + c_b b + c_x x \\ &s.t.\ W_q \leq W_u \\ &B \geq B_l. \end{aligned} \tag{4.2}$$

4.2 Benders' Decomposition Approach

4.2.1 Background

Benders' decomposition was originally developed for solving mixed-integer programming problems [11]. More generally, the method can be applied to any problem in which the variables and constraints can be separated into a master problem and a sub-problem, which are solved in an alternating fashion until an optimality criterion is satisfied. Each time the sub-problem is solved, a constraint which eliminates at least the current solution is added to the

master problem. This ensures that all further solutions either result in better objective function values or are closer to being feasible for the overall problem. The master problem and the sub-problem may be modelled and solved using linear or integer programming [11], or CP, as in logic-based Benders' decomposition [49, 91].

4.2.2 Benders' Decomposition for Optimal Staff Mix

Given the work in Chapter 3, it is natural to decompose our problem into the master problem of finding a combination of cross-trained and specialized workers (a queueing design problem) and the sub-problem of finding a switching policy that satisfies the back room and front room service level constraints given an employee configuration (a queueing control problem).

Our approach is to, firstly, derive a set of constraints on the values of x, f and b by solving problem (4.2) with $x = 0$. These constraints are then used to form the master problem, which is solved in order to identify a cost-optimal, but possibly infeasible, combination of cross-trained and specialized workers, say $x = x'$, $f = f'$ and $b = b'$. If a feasible sub-problem solution can be found, then the master solution is optimal. Otherwise, the cut ($x > x' \;||\; f > f' \;||\; b > b'$) is added to the master problem.[2] The master problem and the sub-problem are then re-solved in order to determine if a feasible policy exists for the new master problem solution.

4.2.3 The "Specialized-Only" Solution

When $x = 0$, the number of workers required for the back room is independent of the number of workers required for the front room. Thus, to find the minimum-cost specialized-only solution, one can independently determine F_{total}, the smallest number of specialized front room workers sufficient to satisfy the waiting time constraint, and B_{total}, the smallest number of specialized back room workers needed to satisfy the back room constraint.

[2]The symbol "||" represents a "logical or". In other words, the cut states that at least one of x, f, b has to be assigned a greater value than its current one in order to achieve feasibility of the sub-problem.

In order to find F_{total}, W_q is calculated for each value of $f \geq 1$ using the equation

$$W_q = \frac{P_0 \sum_{i=1}^{i=S} (\frac{\lambda}{\mu})^{i-1} \frac{i}{D(i)}}{\mu[1 - (\frac{\lambda}{\mu})^S P_0 \frac{1}{D(S)}]} - \frac{1}{\mu}, \tag{4.3}$$

where

$$D(i) = \prod_{j=1}^{i} d(j), \qquad d(i) = \begin{cases} i & \text{if } i \leq f \\ f & otherwise \end{cases} \tag{4.4}$$

and

$$P_0 = 1 + \sum_{i=1}^{i=S} \left(\frac{\lambda}{\mu}\right)^i \frac{1}{D(i)}, \tag{4.5}$$

until the constraint $W_q \leq W_u$ is satisfied for some value of f. We derived this equation from Equation (4.14) using the fact that, when only specialized workers are considered, the front room is a queue with exponential inter-arrival and service times, f servers and a capacity of S (an M/M/f/S queue), in which the probabilities are defined as $P(j) = \left[\left(\frac{\lambda}{\mu}\right)^j/(j!)\right] P_0$ for $1 \leq j < f$ and $P(j) = \left[\left(\frac{\lambda}{\mu}\right)^j/(f!f^{j-f})\right] P_0$ for $f \leq j \leq S$ [42].

B_{total} is simply $\lceil B_l \rceil$ because, when there are no cross-trained workers, this is the smallest possible number of back room workers required to complete all of the back room work.

By definition of F_{total} and B_{total}, there have to be at least F_{total} cross-trained and front room workers in the facility in order for the constraint on W_q to be satisfied, and at least B_{total} cross-trained and back room workers in order for the back room constraint to be satisfied. Thus, constraints $f + x \geq F_{total}$ and $b + x \geq B_{total}$ are valid for problem (4.2). In addition, in any feasible solution, the number of specialized front room workers does not ever have to exceed F_{total}, and the number of specialized back room workers does not ever have to exceed B_{total} because these values already satisfy the constraints in their respective rooms, and having more workers would only incur additional cost. Therefore, $F_{total} - 1$ is an upper bound for f and $B_{total} - 1$ is an upper bound for b. $F_{total} + B_{total} - 1$ is an upper bound for x, since employing a greater number of cross-trained workers will result in a solution of greater cost than the cost of the solution with only specialized workers.

Additionally, since one needs at least F_{total} workers in the facility in order to satisfy the

front room constraint, and at least B_{total} workers in the facility in order to satisfy the back room constraint, it is clear that at least $\max(F_{total}, B_{total})$ workers need to be hired. The lower bound on x is 1 since the best solution with $x = 0$ has already been found.

The cost of the specialized-only solution is an upper bound on the cost of the optimal solution. The lower bound on the optimal cost is the maximum of $c_f F_{total}$ and $c_b B_{total}$ since at least the maximum of F_{total} or B_{total} workers will have to be employed in the facility, and the cost of a cross-trained worker is assumed to be greater than or equal to the cost of either of the specialized workers.

4.2.4 Master Problem

Given the constraints derived from the specialized-only solution, the master problem can be stated as

$$
\begin{aligned}
&minimize\ cost = c_f f + c_b b + c_x x \qquad (4.6)\\
&s.t.\ f + x \geq F_{total}\\
&b + x \geq B_{total}\\
&0 \leq f \leq F_{total} - 1\\
&0 \leq b \leq B_{total} - 1\\
&1 \leq x \leq F_{total} + B_{total} - 1\\
&f + b + x \geq max(F_{total}, B_{total})\\
&max(c_f F_{total}, c_b B_{total}) \leq cost \leq c_f F_{total} + c_b B_{total}\\
&cuts,
\end{aligned}
$$

where $cuts$ are constraints that are added to the master problem each time the sub-problem is not able to find a feasible solution. Initially, $cuts$ is the empty set. Further on, the constraints in this set remove the current optimal solution of the master problem because it does not result in a feasible policy. The master problem is re-solved each time a new cut is added, and the

resulting values of f, b and x define the sub-problem. We solve the master problem with a CP model identical to (4.6). As it is a simple minimization problem with three integer variables, finding a solution is trivial.

4.2.5 Solving the Sub-Problem

Given values for f, b and x, the goal of the sub-problem is to find a policy K such that the constraints $W_q \leq W_u$ and $B \geq B_l$ are satisfied. The sub-problem is very similar to Berman et al.'s problem P_1. Therefore, the method we use for solving the sub-problem is a modification of the *PSums*-P_1 Hybrid method proposed in Chapter 3, which combines a CP model with Berman et al.'s heuristic. More specifically, in order to solve the sub-problem, we run a modified version of Berman et al.'s heuristic P_1 followed by a combination of shaving and search.

In this section, we firstly discuss several possible switching policy formulations and provide details regarding the one that is chosen. Further, we present the equivalents of Berman et al.'s expressions for calculating the quantities of interest given that there may be specialized workers in facility. These expressions are used throughout a modified version of Berman's heuristic, which is presented in Section 4.2.5.2. We then present a CP model of the sub-problem, to which we refer as *PSumsSubproblem*, and the details of shaving procedures used in conjunction with this model.

4.2.5.1 Switching Policy

A switching policy for this problem can be defined in the same way as in the work of Berman et al.: a set of switching points k_i for $i = 0, 1, \ldots, x$ so that the number of cross-trained workers in the front room is i whenever there are between $k_{i-1} + 1$ and k_i customers present in the facility. However, due to the assumption that specialized front room workers can be hired, some of the expressions of Berman et al. have to be adjusted. For example, the balance equations have to be divided into two sets: one for calculating the probabilities of having j customers in the front room when $j < f$, and another for calculating these probabilities when $j \geq f$ (i.e. when some

number of cross-trained workers may be present in the front room). This would make the CP model of the sub-problem more complicated than the models used for Berman et al.'s problem P_1 in Chapter 3.

Another possibility is to use the set of dual variables defined in Section 3.1.3, letting w_j represent the total number of (specialized and cross-trained) workers in the front room when there are j customers in the front room. Constraints of the *Dual* model that are expressed in terms of w_j variables could then be used in a model for the sub-problem with very little modification. For example, the balance equations can still be stated as

$$P(j)\lambda \;=\; P(j+1)w_{j+1}\mu \quad \forall j \in \{0, 1, \ldots, S-1\}. \tag{4.7}$$

However, the *Dual* model of Section 3.1.3 also includes constraints that are expressed in terms of the k_i variables. If we use the k_i policy definition proposed above, these constraints will have to be changed in order to take into account the specialized workers. Additionally, in the experiments of Section 3.4, it was observed that the *Dual* model with shaving is less efficient for problem P_1 than the *PSums* model with shaving. One of the likely reasons for this difference in performance is that each iteration of shaving takes a longer amount of time with the *Dual* model due to a larger number of variables that have to be assigned.

Consequently, the policy formulation that we adopt for the sub-problem is different from the two discussed above. It requires very few changes to the expressions of the *PSums* model and those used in Berman et al.'s heuristic in order to make them applicable to the problem involving specialized workers. In particular, we define a policy as a sequence of "switching points," k_i for $i = 0, 1, \ldots, x + f$, with the interpretation that the number of "busy" workers (ones who are currently serving a customer) in the front room is i whenever the number of customers in the front room is between $k_{i-1} + 1$ and k_i. Switching points with index $i < f$ state that as another customer arrives to the front room, the number of specialized workers who are busy increases; switching points with index $i \geq f$ specify when to switch cross-trained workers between the front room and the back room. Switching point k_{x+f} is a constant and is equal to S. Thus, a policy is vector of k_is, $(k_0, k_1, \ldots, k_{x+f})$, such that $k_i < k_{i+1}$ for

$i = 0, 1, \ldots, x + f - 1$, $k_{x+f} = S$, $k_i = i \; \forall i < f$ and $k_i \geq i \; \forall i \geq f$.

For example, if $f = 2$, $x = 1$ and $S = 6$, the policy $(0, 1, 3, 6)$ states that whenever the number of customers is 1, there is 1 busy worker in the front room and whenever the number of customers is 2 or 3, there are 2 busy workers in the front room. When the number of customers becomes $k_2 + 1 = 4$, a cross-trained worker is switched to the front room. The interpretation that when there are 4 customers in the room, there are 3 busy workers, remains valid, even though one of the workers present is cross-trained.

This policy formulation can be seen as a special case of Berman et al.'s policy formulation with the first f switching points always being fixed to their minimum possible values. Therefore, the equivalent of Theorem 2.4.1, stated below as Theorem 4.2.1, holds for this problem.

Theorem 4.2.1 (Equivalent of Berman et al.'s Theorem 1) *Consider two policies K and K' which are equal in all but one k_i. In particular, suppose that the value of k'_J equals $k_J - 1$, for some J from the set $\{f, \ldots, x + f - 1\}$, $k_J - k_{J-1} \geq 2$, while $k'_i = k_i$ for all $i \neq J$. Then (a) $W_q(K) \geq W_q(K')$, (b) $F(K) \leq F(K')$, (c) $B(K) \geq B(K')$.*

In fact, the set from which J is chosen could be $\{0, \ldots, x + f - 1\}$ since none of the first f components will ever satisfy the condition $k_J - k_{J-1} \geq 2$. Therefore, the only change needed to transform Theorem 2.4.1 into Theorem 4.2.1 is the replacement of N by $x + f$. The proof of Theorem 4.2.1 is the same as that of Theorem 2.4.1, with every occurrence of N in the proof replaced by $x + f$.

As a result of Theorem 4.2.1, we can define the equivalents of Berman et al.'s policies $\hat{K}$ and $\hat{\hat{K}}$. In particular, for the problem involving three types of workers, the policy yielding the greatest possible values of W_q and B is

$$\hat{\hat{K}} = \{k_0 = 0, \ldots, k_{f-1} = f-1, k_f = S-x, k_{f+1} = S-x+1, \ldots, k_{x+f-1} = S-1, k_{x+f} = S\}.$$

The policy which results in the smallest possible values of W_q and B is

$$\hat{K} = \{k_0 = 0, k_1 = 1, k_2 = 2, \ldots, k_{x+f-1} = x + f - 1, k_{x+f} = S\}.$$

The notions of type 1 component and type 2 component can be defined in a manner similar to Berman et al.'s (see Chapter 2). A *type 1 component* is a k_i with the smallest index i satisfying the condition $k_i - k_{i-1} > 1$ for $f \leq i \leq x + f - 1$. A *type 2 component* is a k_i having the smallest index and satisfying the condition $k_{i+1} - k_i > 1$, for $f \leq i \leq x + f - 1$.

For the remainder of the chapter, we refer to a policy satisfying the constraint $B \geq B_l$ as B-feasible, and a policy satisfying the constraint $W_q \leq W_u$ as W_q-feasible. Similarly, policies violating these constraints are called B-infeasible and W_q-infeasible, respectively.

4.2.5.2 Modified Berman et al. Heuristic

Given the above policy formulation, the sub-problem can be modelled analogously to the way Berman et al. model problem P_1. In particular, in order to calculate the quantities of interest (e.g. W_q), a set of probabilities, $P(j)$ for j between 0 and S, has to be defined. Each $P(j)$ denotes the steady-state probability of there being j customers in the front room. These values have to satisfy the set of balance equations

$$P(j)\lambda = P(j+1)i\mu \quad j = k_{i-1}, k_{i-1}+1, \ldots, k_i - 1 \qquad i = 1, \ldots, x+f. \tag{4.8}$$

Consequently, as stated in Section 2.4.1.1, each probability can be calculated using the following expressions:

$$P(j) \;=\; \beta_j P(k_0), \tag{4.9}$$

where

$$\begin{aligned}
\beta_j \;&=\; \begin{cases} 1 \quad \text{if} \quad j = k_0 \\ \\ \left(\frac{\lambda}{\mu}\right)^{j-k_0} \left(\frac{1}{i}\right)^{j-k_{i-1}} X_i \quad \text{if} \quad k_{i-1}+1 \leq j \leq k_i \quad i = 1, \ldots, x+f \end{cases}, \\
X_i \;&=\; \prod_{g=1}^{i-1} \left(\frac{1}{g}\right)^{k_g - k_{g-1}} \\
&\quad (X_1 \equiv 1), i = 1, \ldots, x+f
\end{aligned} \tag{4.10}$$

and

$$P(k_0)\sum_{j=0}^{S}\beta_j = 1. \tag{4.11}$$

The expected number of cross-trained workers in the front room, F_{cross}, is

$$F_{cross} \;=\; \sum_{i=f+1}^{x+f}\sum_{j=k_{i-1}+1}^{k_i} iP(j), \tag{4.12}$$

where the only changes from Equation (2.8) occur in the lower and upper bounds of the sum: the lower bound of 1 is changed to $f+1$ and the upper bound of N is changed to $x+f$. Since there is the possibility of hiring specialized back room workers, the expression for the total expected number of workers in the back room, B, is $b+x-F_{cross}$.

The expected number of customers in the front room is exactly the same as in Equation (2.10):

$$L \;=\; \sum_{j=k_0}^{S} jP(j), \tag{4.13}$$

and the expression for the expected waiting time in the queue is almost identical to Equation (2.11):

$$W_q \;=\; \frac{L}{\lambda(1-P(k_{x+f}))} - \frac{1}{\mu}. \tag{4.14}$$

Based on the above equations, Theorem 4.2.1 and the definitions of $\hat{K}$ and $\hat{\hat{K}}$, the equivalent of heuristic P_1 for this problem, which will be further referred to as heuristic P_3, can be stated as:

1. Start with $K = \hat{\hat{K}}$.

2. If $B(K) < B_l$, the problem is infeasible. Otherwise, let $imb\_W_q = W_q(K)$ and $imb\_K = K$. Set $J = x+f$.

3. Find the smallest j^* s.t. $f \le j^* < J$ and k_{j^*} is a type 1 component. If no such j^* exists, go to 5. Otherwise, set $k_{j^*} = k_{j^*} - 1$. If $B(K) < B_l$, set $J = j^*$ and go to 5. If $B(K) \ge B_l$, go to 4.

4. If $W_q(K) < imb\_W_q$, let $imb\_W_q = W_q(K)$ and $imb\_K = K$. Go to 3.

5. Find the smallest j^* s.t. $f \leq j^* < J$ and k_{j^*} is a type 2 component. If no such j^* exists, go to 6. Otherwise, set $k_{j^*} = k_{j^*} + 1$. If $B(K) < B_l$, repeat 5. Otherwise, go to 4.

6. Stop and return imb_K as the best solution.

This heuristic alternates between trying to reach a policy with smaller W_q and a policy with higher B. Each time a policy which is infeasible for the back room constraint is found, the set of switching points that can be increased or decreased at subsequent steps is reduced in order to prevent cycling. The heuristic stops when it is unable to find any more switching points to decrease or increase, in which case it returns the B-feasible policy with the best value of W_q that it has been able to find.

As stated above, we use a method similar to the *PSums*-P_1 Hybrid to solve the sub-problem. In this method, we run heuristic P_3 first. If the W_q value returned by P_3 is smaller than W_u (the policy is W_q-feasible), then the current sub-problem is feasible[3], and the current master solution is optimal. Otherwise, the CP method based on the *PSumsSubproblem* model and shaving is applied.

4.2.5.3 The *PSumsSubproblem* Model

The *PSumsSubproblem* model has, as its decision variables, the set of k_i, $i = 0, 1, \ldots, x + f$, each with an initial domain of $[0..S]$. Additionally, it has two types of probability variables: $PSums(k_i)$ for $i = 0, ..., x + f - 1$, and $P(j)$ for $j = k_0, k_1, k_2, ..., k_{x+f}$, which are necessary for the calculation of W_q and B given a switching policy. We relate these three types of variables via a set of constraints which ensure that the probability values satisfy the steady-state conditions of the front room queue. We also include constraints for expressing W_q and B in terms of the variables k_i, $PSums(k_i)$ and $P(k_i)$. These constraints are analogous to ones used

[3]There may exist other feasible solutions for the sub-problem, possibly with a smaller W_q than that of the heuristic solution.

in the *PSums* model for problem P_1 (see Section 3.1.2.4).

$PSums(k_i)$ represents the probability of there being between k_i and $k_{i+1} - 1$ customers (inclusive) in the front room and is defined in Equation (4.15). Equation (4.16) is a recursive formula for computing $P(k_{i+1})$, the probability of having k_{i+1} customers in the facility, where $P(k_0) = (\sum_{i=0}^{x+f} \beta Sum(k_i))^{-1}$ and $\beta Sum(k_i)$, $\forall i$, is defined in Equation (4.17).

$$\begin{aligned} PSums(k_i) &= \sum_{j=k_i}^{k_{i+1}-1} P(j) \\ &= \begin{cases} P(k_i)\dfrac{1-\left[\dfrac{\lambda}{(i+1)\mu}\right]^{k_{i+1}-k_i}}{1-\dfrac{\lambda}{(i+1)\mu}} & \text{if } \frac{\lambda}{(i+1)\mu} \neq 1 \\ P(k_i)(k_{i+1}-k_i) & otherwise \end{cases} \end{aligned} \tag{4.15}$$

$$P(k_{i+1}) = \left[\frac{\lambda}{(i+1)\mu}\right]^{k_{i+1}-k_i} P(k_i) \tag{4.16}$$

$$\begin{aligned} \beta Sum(k_i) &= \sum_{j=k_{i-1}+1}^{k_i} \beta_j \\ &= \begin{cases} X_i \left(\dfrac{\lambda}{\mu}\right)^{k_{i-1}-k_0+1} \left(\dfrac{1}{i}\right) \left[\dfrac{1-\left(\dfrac{\lambda}{i\mu}\right)^{k_i-k_{i-1}}}{1-\left(\dfrac{\lambda}{i\mu}\right)}\right] & \text{if } \frac{\lambda}{i\mu} \neq 1 \\ X_i \left(\dfrac{\lambda}{\mu}\right)^{k_{i-1}-k_0+1} \left(\dfrac{1}{i}\right) (k_i - k_{i-1}) & otherwise. \end{cases} \end{aligned} \tag{4.17}$$

The expected total number of cross-trained workers in the front room is

$$F_{cross} = \sum_{i=1}^{x} i[PSums(k_{i+f-1}) - P(k_{i+f-1}) + P(k_{i+f})]. \tag{4.18}$$

The expected number of workers in the back room, B, is defined as $b + x - F_{cross}$.

The expected number of customers in the front room, L, is defined as

$$L = \sum_{i=0}^{x+f-1} L(k_i) + k_{x+f} P(k_{x+f}) \tag{4.19}$$

where

$$L(k_i) = k_i PSums(k_i) + P(k_i)\frac{\lambda}{(i+1)\mu} \times \left[\frac{\left(\frac{\lambda}{(i+1)\mu}\right)^{k_{i+1}-k_i-1}(k_i - k_{i+1}) + \left(\frac{\lambda}{(i+1)\mu}\right)^{k_{i+1}-k_i}(k_{i+1} - k_i - 1) + 1}{\left(1 - \frac{\lambda}{(i+1)\mu}\right)^2} \right].$$

Expected customer waiting time is defined as in Equation (4.14). The derivations of equations (4.15)–(4.19) are not presented in this section as they are almost identical to the derivations of the equivalent expressions in the *PSums* model, Equations (3.6), (3.8)–(3.10) and (3.17).

The equations defining $PSums(k_i)$, $P(k_i)$, L, B and W_q, together with $W_q \leq W_u$ and $B \geq B_l$ are the major constraints of the CP model of the sub-problem. Additionally, the set of constraints necessary for the definition of a policy (presented in Section 4.2.5.1) is included in this model.

4.2.5.4 Shaving

As stated in Sections 2.1.1.3 and 3.3, shaving is a consistency-enforcing procedure for constraint programs based on temporarily adding constraints to the problem, performing propagation and making inferences according to the resulting state of the problem [27, 96]. We use two shaving procedures: $B_l Shaving$, which makes inferences based on the feasibility of policies with respect to the B_l constraint, and $W_u Shaving$, which makes inferences based on feasibility with respect to the W_u constraint. These are similar to the B_l-based and W_q-based shaving procedures used for solving problem P_1, which are presented in Chapter 3 as Algorithms 1 (Figure 3.1) and 2 (Figure 3.2).

Let $\min(k_i)$ and $\max(k_i)$ be, respectively, the smallest and largest values in the current domain of variable k_i, and suppose that the constraint $W_q \leq W_u$ is temporarily removed. At each step of the $B_l Shaving$ procedure, $k_i = \min(k_i)$ or $k_i = \max(k_i)$ is temporarily added to the model for $i \in \{0, ..., x + f - 1\}$. If $k_i = \min(k_i)$ is added, then all other switching points are assigned the maximum possible values subject to the condition that $k_n < k_{n+1}$,

$\forall n \in \{0, ..., x + f - 1\}$. This is done using the function *gMax*, which, given an array of variables, assigns the maximum possible values to all of the variables that do not yet have a value, returning true if the resulting assignment is B-feasible, and false otherwise. If *gMax* returns false and $\min(k_i) + 1 \leq \max(k_i)$, the constraint $k_i > \min(k_i)$ can be permanently added: if all variables except k_i are set to their maximum values, and the problem is infeasible with respect to the B_l constraint, then, by Theorem 4.2.1, in any feasible policy k_i must be greater than $\min(k_i)$. If *gMax* returns false and $\min(k_i) + 1 > \max(k_i)$, there can be no B-feasible policy, and the sub-problem is proven to be infeasible. Otherwise, if *gMax* returns true, we check if the policy is W_q-feasible, in which case the procedure stops since feasibility of the sub-problem has been proved.

If $k_i = \max(k_i)$ is added, the rest of the variables are assigned the minimum values from their domains using the function *gMin*. These assignments are made in a way that respects the constraints $k_n < k_{n+1}, \forall n \in \{0, ..., x + f - 1\}$. If the resulting policy is B-feasible but W_q-infeasible, the constraint $k_i < \max(k_i)$ can be permanently added, assuming $\max(k_i) - 1 \geq \min(k_i)$. Since all variables except k_i are at their minimum values already, and k_i is at its maximum, it must be true, again by Theorem 4.2.1, that in any solution with smaller W_q the value of k_i has to be smaller than $\max(k_i)$. If $\max(k_i) - 1 < \min(k_i)$, there can be no policy with a better W_q than any B-feasible policy encountered so far during the procedure, and the sub-problem is proved to be infeasible. On the other hand, if the policy obtained by applying *gMin* is both B-feasible and W_q-feasible, the procedure stops, and the current master and sub-problem solutions are optimal for our staff mix problem. The complete $B_l Shaving$ algorithm is presented in Figure 4.1.

The $W_u Shaving$ procedure makes inferences based strictly on the constraint $W_q \leq W_u$. The constraint $B \geq B_l$ is removed prior to running this procedure. A constraint of the form $k_i = \max(k_i)$ is added, and the smallest possible values are assigned to the rest of the variables using the function *gMin*. As the B_l constraint has been removed, the only reason why the policy could be infeasible is because it has a W_q value greater than W_u. Since all switching

Algorithm 3: $B_lShaving$
Input: S, μ, λ, B_l, W_u (instance parameters); x, f, b (current master problem solution)
Output: W_q, $policy$ (satisfying policy and its objective value), or (possibly) modified domains of the variables k_i, or proof that sub-problem is infeasible

$changed = true$
while ($changed$)
$changed = false$

for all i from 0 to $x + f - 1$

$successfulShave = true$
while ($successfulShave$)
$successfulShave = false$
add($k_i = \max(Domain(k_i))$)
if (*gMin*)
if ($W_q \leq W_u$)
return $policy$, W_q; stop, sub-problem solution found
if ($\max(Domain(k_i)) - 1 \geq \min(Domain(k_i))$)
add($k_i < \max(Domain(k_i))$)
$successfulShave = true$
$changed = true$
else
stop, no policy with a better W_q value exists; sub-problem is infeasible
remove($k_i = \max(Domain(k_i))$)

$successfulShave = true$
while ($successfulShave$)
$successfulShave = false$
add($k_i = \min(Domain(k_i))$)
if (*gMax*)
if ($W_q \leq W_u$)
return $policy$, W_q; stop, sub-problem solution found
else
if ($\min(Domain(k_i)) + 1 \leq \max(Domain(k_i))$)
add($k_i > \min(Domain(k_i))$)
$successfulShave = true$
$changed = true$
else
stop, no policy with a better W_q value exists; sub-problem is infeasible
remove($k_i = \min(Domain(k_i))$)

Figure 4.1: $B_lShaving$ algorithm

Algorithm 2: $W_u Shaving$

Input: S, μ, λ, B_l, W_u (instance parameters); x, f, b (current master problem solution)

Output: (possibly) modified domains of the variables k_i, or proof that sub-problem is infeasible

$changed = true$
while ($changed$)
  $changed = false$
  for all i from 0 to $x + f - 1$
    $successfulShave = true$
    while ($successfulShave$)
      $successfulShave = false$
      add($k_i = \max(Domain(k_i))$)
      if (!*gMin*)
        if ($\max(Domain(k_i)) - 1 \geq \min(Domain(k_i))$)
          add($k_i < \max(Domain(k_i))$)
          $successfulShave = true$
          $changed = true$
        else
          stop, no policy with a better W_q value exists; sub-problem is infeasible
      remove($k_i = \max(Domain(k_i))$)

Figure 4.2: $W_u Shaving$ algorithm

points except k_i are assigned their smallest possible values, this implies that in any solution with a better expected waiting time, the value of k_i has to be strictly smaller than $\max(k_i)$. If $\max(k_i) - 1 < \min(k_i)$, infeasibility of the sub-problem is proven. The $W_u Shaving$ algorithm is stated in Figure 4.2.

In order to solve the sub-problem, we run $B_l Shaving$ and $W_u Shaving$ in an alternating fashion because the domain reductions inferred during one procedure may lead to further inferences being made by the other. This method stops when a policy satisfying both constraints is found, when a constraint is inferred that violates the current upper or lower bound of a k_i, or when no further inferences can be made. In the final case, standard CP search is performed to determine whether a feasible policy exists.

4.2.6 Summary

Recall that we take a Benders' decomposition approach to our problem of finding the optimal staffing configuration. The master problem is presented as Equation (4.6). As it is a simple minimization problem, it can be easily solved by CP. Once a master solution is obtained, it is used to define the sub-problem, where the aim is to find a policy that would simultaneously satisfy the constraints $B \geq B_l$ and $W_q \leq W_u$. Every time it is proven that no such policy exists for the current master solution, $x = x'$, $f = f'$ and $b = b'$, the cut $(x > x' \;||\; f > f' \;||\; b > b')$ is added to the master problem, and it is re-solved. The alternation between the master problem and the sub-problem continues until a policy is found in the sub-problem which is both B-feasible and W_q-feasible, implying that the current master problem solution is optimal.

The sub-problem is solved by first applying heuristic P_3. If no B-feasible policy exists, the heuristic is able to recognize this, proving the infeasibility of the current master solution. On the contrary, if the heuristic finds a policy which satisfies the back room constraint, two cases may occur. Firstly, the policy may also be W_q-feasible, in which case the current master solution is proven to be optimal. Secondly, the policy may not be able to satisfy the W_q constraint. In this case, there is no guarantee that the sub-problem is infeasible, and we therefore use the

PSumsSubproblem model with shaving and search. Shaving is composed of two procedures, $B_lShaving$ and $W_uShaving$, which are run iteratively until either feasibility[4] or infeasibility of the current master solution is proven, or no more domain reductions are possible. In the latter case, standard CP search is applied to search for a feasible solution or show that none exists.

4.3 Experimental Results and Analysis I

Since our problem has not been previously solved, our first set of experiments focuses on determining some of the reasons for the CPU times required to solve various problem instances and on evaluating the effect of different parameter values on the efficiency of our method. Our approach was implemented in ILOG Solver 6.2, and all experiments were performed on a Dual Core AMD 270 CPU with 1 MB cache, 4 GB of main memory, running Red Hat Enterprise Linux 4.

4.3.1 Problem Size

Recall that S is the maximum total number of customers allowed in the system at any one time and therefore is the prime determinant of problem size. In our first experiment, we use a set of 300 instances, 30 for each value of S from $\{10, 20, \ldots, 90, 100\}$, with costs $c_x = 32$, $c_f = 31$ and $c_b = 30$. The values of the rest of the parameters are taken from the experiments of Chapter 3, for which they were generated in such a way as to ensure non-trivial solutions for Berman's problem P_1. The best W_q values found in those experiments were used as the W_u values for our instances. Since P_1 is similar to our sub-problem, it was expected that using these parameters would give us instances of various difficulty.

Our method solved all but one instance (at S of 80) within 10 CPU minutes. In Table 4.1, the mean CPU time, mean number of times the sub-problem is solved (number of iterations)

[4]Recall that the first master solution that is proven to be feasible is optimal.

Statistic/Value of S	10	20	30	40	50	60	70	80	90	100
CPU Time (seconds)	0.04	0.70	2.59	0.28	0.72	0.55	0.33	93.27	5.25	6.43
# of Iterations	4.57	7.77	16.07	6.13	8.00	7.23	8.23	34.73	27.60	23.83
Total # of Workers	4.07	6.33	9.17	4.80	5.40	5.60	5.27	15.33	8.83	8.93
% Difference Compared to Spec.-only	15.40	4.17	0.48	5.21	5.54	1.28	2.91	0.09	0	0
% Difference Compared to Cross.-only	2.11	2.95	3.71	4.43	4.08	5.72	6.10	5.73	5.90	6.00

Table 4.1: Mean CPU time (seconds), mean number of iterations, mean total number of workers in the optimal solution, and mean percentage differences between the cost of the optimal solution and the two 'naive' solutions for each value of S.

and the mean total number of workers in the optimal solution over 30 instances for each value of S are presented.[5] For the unsolved instance at $S = 80$, we assume, in these calculations, that the run-time is 600 seconds, the number of iterations is the number that was completed within the time limit and the optimal configuration consists of F_{total} front room workers and B_{total} back room workers (this solution is determined in all cases in a negligible amount of time). Therefore, our mean run-time and mean number of iterations statistics are slight underestimates of the true values for S of 80. The total number of workers in the optimal solution statistic may be exact if the optimal solution in the unsolved instance is indeed the specialized-only solution; otherwise, it is a slight overestimate. Similarly, the percentage differences may be exact if the optimal configuration of this instance does not consist of any cross-trained workers.

The mean run-times show a significant peak at $S = 80$ and are approximately correlated with both the total number of workers and the number of iterations. This is not surprising: the more workers are needed in the facility, the more staff combinations usually need to be examined. An increase in the total number of employees in the optimal solution also leads to higher run-times for each sub-problem. This is because the higher the total number of workers in the facility, the larger the size of the policies, and the longer shaving and search will take to prove feasibility or infeasibility. In our problem set, when $S = 80$, there are several instances which have both a large number of iterations (a maximum of 141) and difficult sub-problems (the maximum CPU time required to solve a sub-problem is approximately 82.84 seconds).

Although higher values of S lead to larger domain sizes for the k_is, and consequently, may result in higher run-times, Table 4.1 shows that S is not the main parameter determining the difficulty of a problem instance. In particular, higher values of S do not necessarily lead to higher run-times: the mean CPU time at S of 30 is higher than those at S of 40, 50, 60 and 70, and the mean CPU times at S of 90 and 100 are substantially lower than that at S of 80.

[5]Results are quite sensitive to the level of precision that is set. For example, with a different precision level, the number of iterations and the mean run-time, as well as the optimal staffing configuration, may be different for a particular instance. We set the default Solver precision to 0.000001 for all of our experiments.

λ/μ	10			15			20		
$(\lambda W_u)/B_l$	55.83	21.87	6.15	38.34	10.84	3.55	28.56	8.12	2.37
CPU Time (seconds)	0.01	12.34	12.10	0.01	7.91	38.46	39.92	58.25	91.28
# of Iterations	1.00	3.20	11.30	1.00	5.70	22.58	3.20	14.10	36.50
Total # of Workers	11.00	11.60	12.70	16.00	17.00	18.82	21.60	22.80	25.10
% Difference Compared to Spec.-only	5.27	1.42	0.75	3.69	3.41	1.03	0.75	0.34	1.44
% Difference Compared to Cross.-only	61.46	55.22	46.85	67.92	59.35	48.36	68.01	60.26	47.61

Table 4.2: Mean CPU time (seconds), number of iterations, total number of workers in the optimal solution, and percentage differences between the cost of the optimal solution and the two 'naive' solutions for each value of $\frac{\lambda W_u}{B_l}$.

4.3.2 Cost Combinations

Using the above instances, five different cost combinations, (c_x, c_f, c_b), are examined. When the costs are (32, 1, 31), (32, 31, 1) and (32, 24, 24), all 300 instances are solved, with a mean run-time of under 1 second. For cost combinations (32, 30, 31) and (32, 32, 32), 298 and 295 instances, respectively, are solved in each set, and the mean run-times are 8.16 and 20.84 seconds, respectively. These results indicate that as the difference between the cost of a cross-trained worker and the sum of the costs of specialized workers increases, the time needed to solve the instance increases. In these cases, the master problem is likely to consider solutions with a larger number of cross-trained workers, which implies larger domains for the k_is and longer shaving and search times. The general pattern in the mean CPU times for each S seen in Table 4.1 also occurs under these five cost combinations, indicating that problem difficulty is a function of more than just the cost assumptions.

4.3.3 Other Parameters

In order to examine the effect of the rest of the problem parameters, we use a set of instances with the S, λ, μ, B_l and W_u values from the 54 instances used in the work of Berman et al. [15] (see Section 2.4.2 for more details regarding this set of instances). Ten different cost combinations which satisfy our cost assumptions are used, giving us 540 instances.[6] Although the values of the parameters vary, these instances can be grouped into 9 types according to the value of $\frac{\lambda W_u}{B_l}$, which is an indication of how difficult it will be to satisfy the W_u and B_l constraints simultaneously, adjusted by the arrival rate. Smaller values of this ratio lead to tighter problem instances. In addition, each triple out of these 9 types has an equal value of λ/μ, which is a representation of the workload of the facility.

All 540 instances were solved within 10 minutes. From Table 4.2, it can be seen that as

[6]The cost combinations (c_x, c_f, c_b) used in this experiment are (32, 5, 30), (32, 15, 25), (32, 15, 30), (32, 25, 15), (32, 25, 25), (32, 25, 30), (32, 30, 5), (32, 30, 15), (32, 30, 25) and (32, 30, 30). In several instances, the value of λ or μ had to be rounded from the value used by Berman et al. due to numerical instability.

λ/μ increases, mean CPU times increase. This happens simultaneously with an increase in the mean number of iterations and an increase in the total number of workers in the optimal solution, confirming the observations made from our experiment with different S values. As the value of $\frac{\lambda W_u}{B_l}$ decreases, while λ/μ is held constant, mean CPU times, number of iterations and total number of workers in the optimal solution increase. Therefore, as the workload of the facility increases and/or the expected waiting time bound becomes tighter, the problem becomes harder to solve.

4.3.4 Cost of the Optimal Solution

Both Table 4.1 and 4.2 present the mean percentage difference between the cost of the optimal solution, and the costs of the specialized-only solution and the cross-trained-only solution[7] (i.e. with $f = b = 0$). We calculate the difference between each pair of solutions as a percentage of the cost of the optimal solution. In most cases, there is a non-zero difference between the costs of the optimal solution and the two 'naive' solutions, indicating that our approach may be valuable in practical applications.

4.4 Experimental Results and Analysis II

The sub-problem of our Benders' decomposition method, described in Section 4.2.5, has three main components: the heuristic, the alternating shaving procedure and standard CP search. In this section, we investigate the influence that each of these components has on the effectiveness of our method. To do this, we consider three approaches which are modifications of our original

[7]The cross-trained-only staffing configuration is a solution to Berman's problem P_2 (described in Sections 2.4.2 and 4.1.1). In order to obtain the cross-trained-only solution, we modified our method by changing the upper bound of x to S and the upper bound on the cost to $S \times c_x$ in the master problem (Equation 4.6) and ran it with costs $c_x = 1$, $c_f = 100$, $c_b = 100$. However, our method was unable to solve 19 out of the 300 instances discussed in Section 4.3.1 within 600 seconds. In these cases, we assume that the best cross-trained-only solution consists of $F_{total} + B_{total}$ cross-trained workers. Thus, the percentage differences between the cost of the optimal solution and the cost of the cross-trained-only solution presented in Table 4.1 may be either slight overestimates or slight underestimates. The percentages presented in Table 4.2 are accurate since our modified program was able to find the cross-trained-only solution for all instances within a negligible run-time.

method:

- *heuristicOnly*, in which the solution to the sub-problem can be found only by the heuristic (shaving and search are removed),
- *noShaving*, in which the sub-problem can be solved by either the heuristic or search (the alternating shaving procedure is removed),
- *noSearch*, in which either the heuristic or shaving can solve the sub-problem (search is removed).

The *heuristicOnly* method is a complete method only if, on every iteration in which the heuristic is unable to find a feasible sub-problem solution, the sub-problem is truly infeasible. In particular, the heuristic can guarantee infeasibility of the sub-problem only if the policy $\hat{\hat{K}}$ violates the back room constraint, or the policy $\hat{K}$ violates the front room constraint.

Similarly, the *noSearch* method is complete only when either the heuristic or the alternating shaving procedure is able to guarantee infeasibility on every iteration in which they cannot find a feasible solution. This may happen either when $\hat{K}$ is infeasible with respect to the front room constraint, or $\hat{\hat{K}}$ is infeasible for the back room constraint, or shaving attempts to make a domain reduction that violates the current upper or lower bound of a k_i. If shaving makes some domain reductions but is unable to either find a feasible policy or prove infeasibility, the sub-problem is considered infeasible. Hence, in such cases the *noSearch* method is incomplete.

The *noShaving* method is complete (given enough run-time) because on every iteration in which $\hat{K}$ is feasible for the front room constraint and $\hat{\hat{K}}$ is feasible for the back room constraint, but the heuristic is unable to find a feasible policy, search is run in order to prove the feasibility or infeasibility of the sub-problem. However, if this method takes longer than the allowed run-time limit, we assume that the solution it returns is the specialized-only solution (since this solution is always found within a negligible run-time prior to the execution of the master problem). In these cases, there is no guarantee that this solution is optimal.

We compare the *heuristicOnly*, the *noSearch* and the *noShaving* methods to our original method (which employs a combination of heuristic, shaving and search as discussed in Section 4.2.6) on the set of 540 instances from Section 4.3.3.

4.4.1 Solution Quality

Experiments with our set of 540 instances indicate that the *heuristicOnly* method finds the optimal-cost[8] solution in 442 instances, or in 69.8% of instances for which the optimal is not the specialized-only solution. The *noSearch* method is able to find the optimal-cost solution in 516 instances, or in 92.6% of instances for which the optimal is not the specialized-only solution. The *noShaving* method finds the optimal-cost solution in 389 instances, or 53.5% of cases for which the optimal staffing configuration is not the specialized-only one. The original technique employing the heuristic, shaving and search found the optimal-cost staff mix in all 540 instances.

In Table 4.3, the percentage difference between the cost of the optimal solution (found by our original method) and the costs of the solutions found by the other three methods is presented. It can be observed that the heuristic performs well, but that there exist instances in which it is unable to find the optimal solution. In fact, the table indicates that the *heuristicOnly* method finds the optimal-cost solution in a greater number of instances than the *noShaving* method. This observation can be explained by the fact that the *noShaving* method takes a long time to prove the infeasibility of each infeasible sub-problem, reaches the time limit without having solved the problem in many instances (see the high run-times of the *noShaving* method presented in Table 4.4) and, thus, returns the specialized-only solution in many cases in which this solution is not optimal. The *noSearch* method performs better than either of the other two methods, finding the optimal-cost solution in all cases except when the ratio of λW_u and B_l is 21.87.

[8]The actual staffing configuration found by this method may be different from the one provided by our original method.

λ/μ	10			15			20		
$\frac{(\lambda W_u)}{B_l}$	55.83	21.87	6.15	38.34	10.84	3.55	28.56	8.12	2.37
heuristic	0	0.74	0.24	0	0	0.05	0.22	0.05	0.10
noSearch	0	0.24	0	0	0	0	0	0	0
noShaving	0	1.42	0.75	0	2.20	1.03	0	0.34	1.44

Table 4.3: Mean percentage difference between the cost of the solution provided by the *heuristicOnly*, the *noSearch* and the *noShaving* methods, and the cost of the optimal solution found by our original method for each value of $\frac{\lambda W_u}{B_l}$.

Based on these observations, we can conclude that, in most cases, the optimal-cost solution in our original method is found by either the heuristic or the shaving procedure, indicating that both are essential components of our overall approach.

4.4.2 Mean Run-time

Table 4.4 presents the mean run-times for the *heuristicOnly*, the *noSearch* and the *noShaving* methods, and our original method. The only method out of these four that is unable to solve some instances within 600 seconds is the *noShaving* method (in fact, it does not solve 242 instances). In the calculation of the mean run-times for this method, we assume that the run-time of instances that are not solved is 600 seconds, and hence the mean run-time figures for the *noShaving* method presented in the table are under-estimates of the true means.

We can, firstly, observe that the mean run-times of our original method most closely resemble those of the *noSearch* method. Secondly, it can be seen that the *heuristicOnly* method is much faster than the rest, with mean run-times being below two seconds for all values of $\frac{\lambda W_u}{B_l}$. This is not surprising, since the original heuristic of Berman et al. has been shown in Section 3.4.2 to be extremely fast in our experiments with problem P_1.

Thirdly, the longest run-times result from the *noShaving* method, since search has to be

λ/μ	10			15			20		
$\frac{(\lambda W_u)}{B_l}$	55.83	21.87	6.15	38.34	10.84	3.55	28.56	8.12	2.37
heuristic	0.01	0.07	0.26	0.01	0.18	0.82	0.05	0.60	1.78
noSearch	0.01	8.12	12.07	0.01	7.94	38.49	39.84	58.11	91.45
noShaving	0.01	249.87	421.81	0.01	358.97	500.00	193.04	430.02	540.00
original	0.01	12.34	12.10	0.01	7.91	38.46	39.92	58.25	91.28

Table 4.4: Mean run-times for *heuristicOnly*, *noSearch*, *noShaving* and our original method for each value of $\frac{\lambda W_u}{B_l}$.

used in all cases for which the heuristic is unable to guarantee infeasibility and also unable to find a feasible solution. In Section 3.6.1, we stated that one possible reason for the *PSums* model without shaving being unable to prove optimality in some instances of problem P_1 is that there is very little propagation from the constraint $W_q \leq bestW_q$ (where $bestW_q$ is the best value of the expected waiting time found up to that point in the search). The *PSums* model served as the basis of the sub-problem model, and the two are similar in structure. Therefore, we can conjecture that one of the reasons for the search taking a long time to prove or disprove the feasibility of the sub-problem in some instances is the lack of propagation from the constraint $W_q \leq W_u$.

These observations indicate that a majority of problems can be solved by using the heuristic and shaving, and that, in such cases, most of the run-time of our original method is spent in the shaving stage. More importantly, it is clear that the use of shaving greatly reduces the run-times needed by our original method to guarantee optimality. We can also conjecture that, in general, one of the reasons why there exist instances in which our original method is unable to solve the problem within 600 seconds (i.e. some instances at S of 80 in experiments of Sections 4.3.1 and 4.3.2) is the fact that shaving is unable to prove the feasibility or infeasibility of the last master problem solution considered and search needs a long time to do so.

λ/μ	10			15			20		
$(\lambda W_u)/B_l$	55.83	21.87	6.15	38.34	10.84	3.55	28.56	8.12	2.37
heuristicOnly	1.00	4.50	12.20	1.00	5.70	22.93	3.90	14.60	38.00
noSearch	1.00	3.60	11.30	1.00	5.70	22.58	3.20	14.10	36.50
noShaving	1.00	3.20	3.60	1.00	2.70	4.37	3.20	3.60	6.20
original	1.00	3.20	11.30	1.00	5.70	22.58	3.20	14.10	36.50

Table 4.5: Mean number of iterations for *heuristicOnly*, *noSearch*, *noShaving* and our original method for each value of $\frac{\lambda W_u}{B_l}$.

4.4.3 Mean Number of Iterations

In Table 4.5, we see that the mean number of iterations for all but one value of $\frac{\lambda W_u}{B_l}$ is the same for our original method and the *noSearch* method, which is not surprising given the similarity in the mean run-times between the two. The slight difference between the two methods for $\frac{\lambda W_u}{B_l} = 21.87$ is due to the fact that, as shown in Section 4.4.1, this subset of instances is the only one for which the *noSearch* method is unable to find the optimal solution in some cases. In particular, for such instances shaving is not able to prove feasibility of the sub-problem given the optimal master solution, it is (incorrectly) assumed that the sub-problem is infeasible, and the next master solution is considered (increasing the number of iterations). On the other hand, in our original method, search is able to recognize the feasibility of the sub-problem in these cases, resulting in a smaller number of iterations.

We should also note that the *noShaving* method has a smaller or equivalent number of iterations compared to the other methods. This is due to the fact that using search to solve each sub-problem often results in overall run-times of greater than 600 seconds after only a small number of iterations.

The *heuristicOnly* method has the largest number of iterations out of the four methods for all values of λ/μ. This is because the heuristic misses some feasible solutions and, as a result,

has to consider more master solutions, increasing the total number of iterations.

4.4.4 Summary

Overall, these results indicate, firstly, that the alternating shaving procedure is an essential part of our method, as it significantly reduces the run-time required to prove the infeasibility of each infeasible sub-problem, and, therefore, decreases the overall run-times of our method. In fact, without shaving, our method would not have been able to solve a large proportion of our problem instances to optimality. Additionally, shaving may also find a feasible solution in a short amount of time if the heuristic is unable to do so.

Secondly, we see that the heuristic is a helpful component of our method, since, in most instances, it allows us to quickly guarantee the feasibility of the sub-problem when the master problem solution is optimal.

Search is also a useful component of our method, since there are cases when the heuristic and shaving are unable to find a feasible policy for the optimal master solution. However, since the *noSearch* method is able to guarantee optimality in most cases (see Table 4.3), search is seldom used. When it is needed for guaranteeing completeness, it usually results in the instance not being solved within 600 seconds.

In order to summarize these observations, we can say that the most important component of our method is the alternating shaving procedure, but that the other two components also play a significant role in some instances.

4.5 Conclusions

In this chapter, a Benders' decomposition-based constraint programming method is proposed for the problem of determining the optimal mix of cross-trained and specialized workers in a retail facility given specific assumptions regarding the staffing costs of these workers (other cost assumptions are discussed in the following chapter). This method is shown to perform

well over a wide range of problem parameters. In many cases, the cost of the optimal solution is shown to deviate from both the solution in which all workers are specialized and the solution in which all workers are cross-trained. An examination of the method used to solve the sub-problem indicates that all three of its components (the heuristic, the alternating shaving procedure and the search) are important, but that the effectiveness of our method is primarily due to the shaving procedure's ability to prove the infeasibility of each infeasible sub-problem quickly.

Since the problem discussed is a queueing design and control problem, this chapter demonstrates that constraint programming may be a useful approach for solving such problems.

Chapter 5

An Analysis of Cost Cases in the Optimal Staff Mix Problem

In Chapter 4, the problem of finding the optimal staffing configuration in a retail facility is discussed under the assumption that the cost of a cross-trained worker is less than or equal to the sum of the costs of specialized workers, but greater than or equal to each of the specialized worker costs individually. In this chapter, other possible combinations of costs are analyzed. A major issue which arises in this analysis is the formulation of a switching policy. We therefore start with a discussion of two possible policy formulations, both for Berman et al.'s problem P_2 and for the optimal staff mix problem. We then state the implications of these formulations on the solution of the latter problem under different cost assumptions. It is shown that the work presented in Chapter 4 remains valid with the new policy formulation.

5.1 Policy Definitions for Berman et al.'s Problem P_2

In the second problem addressed by Berman et al. [15], P_2, the goal is to find the smallest number of cross-trained workers, N, and a switching policy which would state how to assign these workers to the front and back rooms of a retail facility with capacity S. Recall that

Berman et al. define a switching policy in terms of quantities k_i, for $i = 0, \ldots, N$, so that there are i workers in the front room whenever there are between $k_{i-1} + 1$ and k_i customers in the front room, for $i = 1, 2, \ldots, N$. See Section 2.4.2 for details regarding problem P_2. Given a family of switching policies $\mathbf{K} = \{K; K = \{k_0, k_1, ..., k_{N-1}, S\}$, k_i integers, $k_i - k_{i-1} \geq 1$, $k_0 \geq 0, k_{N-1} < S\}$, the problem can formally be stated as:

$$\begin{aligned} &minimize_{K \in \mathbf{K}}\ N \\ &s.t.\ W_q \leq W_u \\ &B \geq B_l, \end{aligned} \tag{5.1}$$

where W_q is the expected customer waiting time, W_u is the upper bound on the expected customer waiting time, B is the expected number of workers in the back room, and B_l is the minimum expected number of workers in the back room that is required in order for all work to be completed. The values of B and W_q are calculated from a particular switching policy as shown in Equations (2.9) and (2.11), respectively, in Chapter 2.

Berman et al. state that, because minimizing W_q and maximizing B are conflicting goals, it may not be possible to find a number, N, of workers that would allow both constraints to be satisfied. This happens because of the back room constraint $B \geq B_l$. In other words, there exists some value of N for which even the policy $\hat{\hat{K}} = \{k_0 = S-N, k_1 = S-N+1, ..., k_{N-1} = S-1, k_N = S\}$, which results in the greatest possible values of W_q and B, is feasible for the front room constraint but infeasible for the back room constraint. Adding more workers further reduces the expected waiting time, but only marginally increases the B value of the resulting $\hat{\hat{K}}$ policy. Since the size of a policy[1] is N and the size of the domain of each k_i is $S - N + 1$, Berman's policy formulation cannot be used to define policies with $N > S$. Therefore, if it happens that, when $N = S$, the policy $\hat{\hat{K}}$ is infeasible, then there is no feasible solution for this instance of the problem.

[1]The size of a policy is the number of switching points.

N	W_q	B
22	1.29406	0.362632
23	1.19391	0.385196
24	1.10209	0.408486
25	1.01761	0.432537
26	0.939606	0.457384
27	0.86737	0.483067
40	0.255857	0.915678
49	0.0209997	1.38034
50	–	1.44473

Table 5.1: Values of W_q and B for policy $\hat{\hat{K}}$ for various values of N for the policy $\hat{\hat{K}}$. The value of W_q could not be calculated exactly when $N = S = 50$ due to precision issues in the ILOG Solver program used to do so.

As an example, consider an instance of the problem with $S = 50$, $\lambda = 80$, $\mu = 1$, $B_l = 1.92$, $W_u = 1.35$. In Table 5.1, values of B and W_q for various values of N (up to S) are presented. It can be seen that as N increases, the value of W_q decreases, even after it becomes lower than W_u at $N = 22$. The value of B increases with N, but only marginally, and is not feasible when $N = S$.

In practical applications, the infeasibility of this problem does not make sense–clearly, $N = S + \lceil B_l \rceil$ should allow both constraints to be simultaneously satisfied, since every customer will be served immediately, and there will be a sufficient number of workers in the back room. The reason for this inconsistency is the definition of a switching policy. In particular, a policy as it is defined above assumes that all of the cross-trained workers will be switched at some point in time. This definition does not allow a worker to be permanently assigned to the back room and not switch. One may say that assigning a cross-trained worker to only one type of work contradicts the original idea of hiring cross-trained workers. However, if the facility has a policy of hiring cross-trained workers only, the decision to permanently assign some of them to the back room may be necessary in order to ensure that all of the work in the back room is completed.

An alternative policy definition could be used in order to resolve this inconsistency. Given N cross-trained workers, a policy can be represented by a tuple $< K, b^* >$, where K is a switching policy as defined in Berman et al. (stated above) and b^* is a non-negative integer that represents the number of workers that are always assigned to back room work. The size of the policy K is therefore $N - b^* + 1$. In the above example, the policy $< \hat{\hat{K}}, 2 >$ with $N = 24$ is feasible.

This new policy formulation is more realistic but also increases the search space of the problem. In particular, in order to find a feasible value of N, one would have to devise an effective method of searching through tuples $< K, b^* >$ in which $b^* \leq \lceil B_l \rceil$, so as not to enumerate all of them.

5.2 Policy Definitions for the Optimal Staff Mix Problem

Now, consider the extension of Problem (5.1) that was discussed in Chapter 4: the optimal staff mix problem. In this problem, we suppose that the facility can hire specialized front room and specialized back room workers in addition to cross-trained ones, and that each type of worker has a different cost.

As in Chapter 4, we let x denote the number of cross-trained workers, f the number of front room workers, and b the number of back room workers hired by the retail facility. c_x, c_f and c_b denote the costs of a front room worker, a back room worker and a cross-trained worker, respectively. F_{total} is the smallest number of workers needed in the front room in order to achieve the desired quality of service, while B_{total} is the smallest number of workers needed in the back room in order to complete all of the necessary work.

The goal of the problem is to find the lowest-cost combination of workers which would ensure that the expected number of workers in the back room is sufficient to complete all of the necessary work and that the expected customer waiting time is bounded from above. The formal definition of this problem has been stated as Equation (4.2) and is repeated here for convenience in Equation (5.2).

$$\begin{aligned} &minimize\ c_f f + c_b b + c_x x \\ &s.t.\ W_q \leq W_u \\ &B \geq B_l. \end{aligned} \tag{5.2}$$

Given a particular switching policy, B and W_q can be calculated using Equations (4.12) and (4.14), respectively. For a more detailed description of this problem, and a complete listing of constraints, the reader is referred to Chapter 4.

A policy is defined as a vector of k_is, $(k_0, k_1, \ldots, k_{x+f})$, such that $k_i < k_{i+1}$ for $i = 0, 1, \ldots, x + f - 1$, $k_{x+f} = S$, $k_i = i\ \forall i < f$ and $k_i \geq i\ \forall i \geq f$, with the interpretation that the number of "busy" workers (ones who are currently serving a customer) in the front room

is i whenever the number of customers in the front room is between $k_{i-1}+1$ and k_i. We will further refer to this policy definition as the "K-policy" definition. As shown in Chapter 4, this formulation implies that the policy $\hat{\hat{K}} = \{k_0 = 0, k_1 = 1, ..., k_{f-1} = f-1, k_f = S-x, k_{f+1} = S-x+1, ..., k_{x+f-1} = S-1, k_{x+f} = S\}$ yields the greatest possible values of W_q and B, while the policy $\hat{K} = \{k_0 = 0, k_1 = 1, k_2 = 2, ..., k_f = f, ..., k_{x+f-1} = x+f-1, k_{x+f} = S\}$ results in the smallest possible values of W_q and B.

Due to its similarity with the policy definition of Berman et al. used for Problem (5.1), this type of policy does not allow cross-trained workers to be permanently assigned to the back room. This implies, firstly, that a "cross-trained-only" solution, in which $f = 0$ and $b = 0$, may not always exist for this problem. Secondly, this means that any reasoning based on the assumption that, given a staff configuration in which the number of front room workers is sufficient for the satisfaction of the front room constraint, B_{total} cross-trained workers will be sufficient to satisfy the back room constraint, is invalid. Such an assumption holds in practical situations, since, if we know that employing B_{total} specialized back room workers is enough to complete all tasks, and if cross-trained and specialized workers are assumed to be equally skilled at performing back room work, then having B_{total} cross-trained workers should lead to the satisfaction of the back room constraint as well. In order to deal with this discrepancy between a realistic assumption and the consequences of a particular policy formulation, we can alternatively define a policy as a tuple $< K, b^* >$, where K is specified according to the K-policy definition, and b^* is a non-negative integer that represents the number of cross-trained workers that are always assigned to back room work. We will refer to this formulation as the "$< K, b^* >$-policy" definition. Under this formulation, the policy $< \hat{\hat{K}}, \lceil B_l \rceil >$ results in the greatest possible W_q and B, and the policy $< \hat{K}, 0 >$ results in the smallest possible W_q and B. Note that $W_q(< \hat{\hat{K}}, \lceil B_l \rceil >) \geq W_q(\hat{\hat{K}})$ and that $B(< \hat{\hat{K}}, \lceil B_l \rceil >) \geq B(\hat{\hat{K}})$, but $W_q(< \hat{K}, 0 >) = W_q(\hat{K})$ and $B(< \hat{K}, 0 >) = B(\hat{K})$.

Regardless of the policy definition, we will refer to both a policy satisfying the constraint $B \geq B_l$ and a staffing configuration for which it is possible to find such a policy as B_l-feasible.

A policy satisfying the constraint $W_q \leq W_u$ and a staffing configuration for which such a policy can be found will be further called W_u-feasible.

5.3 Five Cost Cases

The difficulty of solving Problem (5.2) depends on the assumptions that are made about the costs of the different workers. These assumptions can be divided into five cost cases:

1. $c_b > c_x > c_f$,
2. $c_f > c_x > c_b$,
3. $c_x \geq c_f + c_b$,
4. $c_x \leq c_f$ and $c_x \leq c_b$,
5. $c_x \leq c_b + c_f$ and $c_x \geq c_f$, $c_x \geq c_b$.

In the next section, we discuss these cases assuming that the K-policy definition is used. In Section 5.5, the changes to these five cost cases resulting from the use of the $< K, b^* >$-policy are presented.

5.4 Cost Cases with the K-Policy Definition

In this section, we analyze the five cost cases under the assumption that the K-policy definition is used. For each cost case, we state what the optimal solution will look like based on the cost assumptions, provide a proof of its optimality, and describe how it may be found.

5.4.1 Cost Case 1: $c_b > c_x > c_f$

If we ignore the feasibility issues discussed above, it is clear that, given these cost assumptions, the optimal solution should only have cross-trained and front room workers. Taking feasibility

issues into account, we know that this is not necessarily the case, since having B_{total} cross-trained workers may not result in any B_l-feasible policies. We can say, however, that if the policy $\hat{\hat{K}}$ with B_{total} cross-trained workers is B_l-feasible, then the optimal solution will have the form $x = B_{total}, f \leq F_{total}, b = 0$. The following claim states this more formally.

Proposition 5.4.1 *If $c_b > c_x > c_f$ and the solution $x = B_{total}, f = 0, b = 0$ is B_l-feasible, then the optimal solution is always of the form $x = B_{total}, f = f', b = 0$, where $f' \leq F_{total}$ and f' is the smallest integer that makes this solution feasible.*

Proof: We start with solution $x = 0, f = F_{total}, b = B_{total}$, which is feasible by definition, and try to construct a smaller-cost feasible solution. Since $c_b > c_x$, substituting each of the B_{total} back room workers by cross-trained ones will result in lower cost. The resulting solution is still W_u-feasible since $f = F_{total}$, and still B_l-feasible since, by assumption, there exists a B_l-feasible policy with B_{total} cross-trained workers.

Since some of the B_{total} cross-trained workers may help to satisfy the front room constraint, the cost of the resulting solution may be further reduced by decreasing the number of front room workers. We therefore reduce f until the smallest possible value, f', that makes the solution $x = B_{total}, f = f', b = 0$ feasible. It is not possible to modify this solution further in order to reduce its cost and still retain its feasibility: x cannot be decreased because of the constraint $b + x \geq B_{total}$, f cannot be decreased because f' is the smallest value for f that makes this solution feasible, and x cannot be increased to replace some specialized front room workers because $c_x > c_f$.

Thus, the solution $x = B_{total}, f = f', b = 0$ is always optimal under the given cost assumptions and the assumption that a B_l-feasible policy exists with B_{total} cross-trained workers. ■

If we remove the assumption that a B_l-feasible policy exists with B_{total} cross-trained workers, then the claim is not true, and the optimal solution may be the specialized-only solution

(i.e. in which $x = 0$), or a solution with some number of cross-trained and some number of specialized workers. In particular, if we start with the specialized-only solution, $x = 0, f = F_{total}, b = B_{total}$, we may be able to substitute *some*, but not all, of the B_{total} back room workers by cross-trained ones and find a smaller-cost feasible solution. However, it may also happen that no feasible solution of smaller cost exists with $b \leq B_{total}$.

Consequently, under the given cost assumptions, the problem can be solved by first checking whether the policy $\hat{\hat{K}}$ with B_{total} cross-trained workers (and no specialized workers) is B_l-feasible. If it is, the above claim is true, and the problem can be solved quite easily by first constraining x to equal B_{total}, b to equal 0, and then decreasing the value of f, starting from F_{total}, until it is not possible to find a feasible policy for the resulting combination of workers. The last value of f for which a feasible policy exists will be the required optimal value, f'. If $\hat{\hat{K}}$ with B_{total} cross-trained workers is not feasible, then the method proposed in Chapter 4 can be applied, with slight modifications. Firstly, the constraint $x \leq F_{total} + B_{total} - 1$ has to be substituted by the constraint $x \leq S$, since there may exist a feasible solution with more than $F_{total} + B_{total} - 1$ cross-trained workers that will give smaller cost than the specialized-only solution. Secondly, the cut ($x > B_{total} \;\; || \;\; b > 0$) can be added to the master problem before the first iteration.

5.4.2 Cost Case 2: $c_f > c_x > c_b$

Because a cross-trained worker is less expensive than a front room one, intuitively, the optimal solution should have $x = F_{total}$, $f = 0$ and $b \leq B_{total}$. Recall that policy $\hat{K}$ (defined in Section 5.2) with F_{total} front room workers is exactly the same as policy $\hat{K}$ with F_{total} cross-trained workers, since in both cases, an additional server becomes busy as a customer arrival occurs. The only difference between the two cases is that if these workers are cross-trained, they are switched from the back room to the front room, whereas if they are specialized, they are "switched" from being idle to being busy. Hence, the two policies yield the same W_q values. Since employing F_{total} front room workers yields a $\hat{K}$ policy which is W_u-feasible (by

definition of F_{total}), employing the same number of cross-trained workers and switching them according to $\hat{K}$ will also lead to the satisfaction of the W_q constraint. Due to these properties, specialized front room and cross-trained workers can be substituted for each other without violating the W_u-feasibility of a particular staffing configuration, which leads to the following proposition.

Proposition 5.4.2 *If $c_f > c_x > c_b$, then the optimal solution is always of the form $x = F_{total}, f = 0, b = b'$, where $b' \leq B_{total}$ and b' is the smallest integer that makes this solution feasible.*

Proof: We start with solution $x = 0, f = F_{total}, b = B_{total}$, which is feasible by definition, and try to construct a smaller-cost feasible solution. Since $c_f > c_x$, substituting each of the F_{total} front room workers by cross-trained ones leads to lower cost. The resulting configuration remains W_u-feasible since policy $\hat{K}$ with $x = F_{total}$ is equivalent to policy $\hat{K}$ with $f = F_{total}$. It is also still B_l-feasible since the value of b has not been changed.

Since some of the F_{total} cross-trained workers may help to satisfy the back room constraint, the cost of the resulting solution may be further reduced by reducing the number of back room workers. We therefore reduce b until the smallest possible value, b', that makes the solution $x = F_{total}, f = 0, b = b'$ feasible. It is not possible to modify this solution further in order to reduce its cost and still retain its feasibility: x cannot be decreased because of the constraint $f + x \geq F_{total}$, b cannot be decreased because b' is the smallest value for b that makes this solution feasible, and x cannot be increased to replace some specialized back room workers because $c_x > c_b$.

Thus, the solution $x = F_{total}, f = 0, b = b'$ is always optimal under the given cost assumptions. ■

The problem of finding the optimal staff configuration given that $c_f > c_x > c_b$ can be solved quite easily by first constraining x to equal F_{total}, f to equal 0, and then decreasing the

value of b, starting from B_{total}, until it is not possible to find a feasible policy for the resulting combination of workers. The last value of b for which a feasible policy exists will be the required optimal value, b'.

5.4.3 Cost Case 3: $c_x \geq c_f + c_b$

In the case when the costs satisfy the assumption $c_x \geq c_f + c_b$, there is no incentive to hire any cross-trained workers, since doing so will always lead to higher-cost configurations. In particular, we know that in any feasible solution, a cross-trained worker can always be substituted by a pair of specialized workers, one for the front room and one for the back room, without violating feasibility. Since such a pair of specialized workers is less expensive than one cross-trained worker, the specialized-only solution, $x = 0, f = F_{total}, b = B_{total}$, is optimal. The following claim and proof state these ideas more formally.

Proposition 5.4.3 *If $c_x \geq c_f + c_b$, then the optimal solution is always $x = 0, f = F_{total}, b = B_{total}$.*

Proof: Suppose there exists a solution of smaller cost than the cost of solution $x = 0, f = F_{total}, b = B_{total}$. Any such solution will have the form $x = x', f = F_{total} - f', b = B_{total} - b'$. There are three cases which may result in a smaller-cost solution:

1. $x' < f'$

   This will be infeasible because $x' + (F_{total} - f') < F_{total}$[2].

2. $x' < b'$

   This will be infeasible because $x' + (B_{total} - b') < B_{total}$.

3. $x' \geq f'$ and $x' \geq b'$

[2]The constraints $x + f \geq F_{total}$ and $x + b \geq B_{total}$ are discussed in Section 4.2.3.

The cost of this solution will be $c_f(F_{total}) + c_b(B_{total}) + c_x(x') - c_f(f') - c_b(b')$, which is greater than or equal to $c_f(F_{total}) + c_b(B_{total})$ because

$$c_x(x') - c_f(f') - c_b(b') \geq c_x(x') - c_f(x') - c_b(x') = x'(c_x - c_f - c_b) \geq 0$$

where the last inequality follows from the assumption that $c_x \geq c_f + c_b$.

Therefore, there is no solution of better cost and $x = 0, f = F_{total}, b = B_{total}$ is optimal (if $c_x = c_f + c_b$, then there may be alternative optimal solutions). ■

Given these cost assumptions, the problem can be solved by finding the values of F_{total} and B_{total}. F_{total} can be found by calculating the value of W_q for each value of f starting from 1 until the constraint $W_q \leq W_u$ is satisfied. The expression for W_q given f front room workers is simply that of the expected waiting time in an M/M/f/S queue [42], or can be calculated using Equation (4.3).

5.4.4 Cost Case 4: $c_x \leq c_f$ and $c_x \leq c_b$

In the fourth cost case, we assume $c_x \leq c_f$ and $c_x \leq c_b$. Under these cost assumptions, every specialized worker in a given staffing configuration can be substituted by a cross-trained one without increasing the total cost of that configuration, implying that the cross-trained-only solution should be optimal. However, similarly to Cost Case 1, the form of the optimal solution is dependent on whether policy $\hat{\hat{K}}$ with solution $x = B_{total}, f = 0, b = 0$ is B_l-feasible. If this is so, then the optimal solution will consist of cross-trained workers only.

Proposition 5.4.4 *If $c_x \leq c_f$ and $c_x \leq c_b$, and the solution $x = B_{total}, f = 0, b = 0$ is B_l-feasible, then the optimal solution is always of the form $x = x', f = 0, b = 0$, where $F_{total} + B_{total} \geq x' \geq \max(F_{total}, B_{total})$ and x' is the smallest number of cross-trained workers that are needed in the facility in order to satisfy both constraints.*

Proof: We start with solution $x = 0, f = F_{total}, b = B_{total}$, which is feasible by definition, and try to construct a smaller-cost feasible solution. Since $c_x \leq c_f$, substituting each of the F_{total} front room workers by cross-trained ones will result in a lower-cost (feasible) solution. Feasibility is guaranteed because the policy $\hat{K}$ with F_{total} cross-trained workers is equivalent to the policy $\hat{K}$ with F_{total} specialized front room workers.

Since $c_x \leq c_b$ and we assume that the solution $x = B_{total}, f = 0, b = 0$ is B_l-feasible, we can substitute all B_{total} back room workers by cross-trained ones, giving us the solution $x = F_{total} + B_{total}, f = 0, b = 0$.

Due to the possibility of switching these workers between the two rooms, we may be able to reduce the value of x until x', the smallest number of cross-trained workers for which a policy that is both B_l-feasible and W_u-feasible can be found.

It is then not possible to find a smaller-cost feasible solution for two reasons. Firstly, substituting any of the cross-trained workers by specialized ones will necessarily result in higher cost due to the cost assumptions. Secondly, decreasing x' will result in infeasibility, since x' is the smallest number of cross-trained workers for which a feasible policy can be found.

Thus, the solution $x = x', f = 0, b = 0$ is always optimal under the given cost assumptions and the assumption that $x = B_{total}, f = 0, b = 0$ is B_l-feasible. ■

If the assumption of the feasibility of solution $x = B_{total}, f = 0, b = 0$ with respect to the back room constraint is violated, then the optimal solution will consist of at least F_{total} cross-trained workers, no specialized front room ones and possibly some number of back room workers. Following the structure of the proof of the above claim, suppose we are given the specialized-only (feasible) solution $x = 0, f = F_{total}, b = B_{total}$. We can substitute all of the front room workers by cross-trained ones without violating feasibility. After this, we may not be able to find a smaller-cost feasible solution by increasing the number of cross-trained workers and decreasing the number of back room workers, since there may not exist a value

of x that results in a B_l-feasible policy. In this case, the optimal solution will have $x \geq F_{total}$, $f = 0$, and $0 \leq b \leq B_{total}$.

Consequently, if $c_x \leq c_f$ and $c_x \leq c_b$, the problem may be solved by first checking the feasibility of solution $x = B_{total}, f = 0, b = 0$ with respect to the B_l constraint. Given this configuration, if policy $\hat{\hat{K}}$ is B_l-feasible, then we can constrain f and b to be 0 and enumerate the possible values of x starting at 1 until a policy can be found for some x which satisfies both constraints. In other words, we have to solve Berman et al.'s problem P_2. Otherwise, the method presented in Chapter 4 can be applied, with the constraint $x \leq B_{total} + F_{total} - 1$ substituted by $x \leq S$, since it is possible for the optimal solution to consist of more than $F_{total} + B_{total}$ cross-trained workers.

5.4.5 Cost Case 5: $c_x \leq c_b + c_f$ and $c_x \geq c_f$, $c_x \geq c_b$

Consider now the cost case that was extensively studied in Chapter 4. Clearly, under the given cost assumptions, hiring a cross-trained worker will be less expensive than hiring a pair of specialized ones. However, if, in order to satisfy both constraints, we need more than one cross-trained worker, it may be cheaper to hire some number of front and back room workers instead. For example, if $c_x = 7$, $c_f = 5$ and $c_b = 4$, hiring a cross-trained worker is less expensive than hiring a pair of specialized ones, but hiring 3 front room workers and 1 back room worker is less expensive than hiring 3 cross-trained workers. Therefore, in this case, it is not clear what the optimal solution will look like. In fact, for these cost assumptions, there are 5 possibilities for the optimal solution:

1. $x = x'$, $f = 0$, $b = 0$ (only cross-trained workers), $1 \leq x' \leq S$

   For example, this will happen if x' is smaller than $F_{total} + B_{total}$ and there is no feasible lower-cost solution $x = x' - x''$, $f = f'$, $b = b'$ with $x'' < f' + b'$ (i.e. when hiring more workers cannot result in a feasible smaller-cost solution, which could happen in the above example with costs $c_x = 7$, $c_f = 5$ and $c_b = 4$).

2. $x = 0$, $f = F_{total}$, $b = B_{total}$ (only specialized workers)

   This will happen when all solutions of smaller cost with $b < b'$ are infeasible.

3. $x = x'$, $f = f'$, $b = 0$ (no back room workers), $f' \leq F_{total}$, $x' > 0$

   For example, this will happen if the expected number of required back room workers is fractional, and both constraints can be satisfied by hiring some number of cross-trained workers rather than by hiring pairs of specialized back room and front room workers.

4. $x = x'$, $f = 0$, $b = b'$ (no front room workers), $b' \leq B_{total}$, $x' > 0$

   For example, if, given some number of back room workers, there needs to be a worker in the front room and another one in the back room for only a fraction of the time, then it may be reasonable to hire a cross-trained worker instead of a pair of specialized ones.

5. $x = x'$, $f = f'$, $b = b'$, $b' < B_{total}$, $f' < F_{total}$, $x' > 0$

   A solution of this kind may be optimal if the expected number of workers required in the back room and front room is fractional.

Because the optimal solution in this case may take any of the five forms listed, it is also harder to find than in the other cost cases. A detailed discussion of a method for solving this problem has been presented in Chapter 4.

5.5 Cost Cases with the $< K, b^* >$-Policy Formulation

The alternative policy definition provided in Section 5.2 implies that a cross-trained-only feasible solution always exists. We can find such a solution by enumerating the value of x starting from $x = 1$ until the smallest possible value of x is found that results in a feasible $< K, b^* >$ policy. This value will be further referred to as X_{total}, and any solution with $f = 0$ and $b = 0$ will be called the "cross-trained-only" solution. The most important consequence of the change in the policy formulation is that X_{total} becomes a lower bound on the total number of workers in the facility, as stated in the following theorem.

Theorem 5.5.1 (Lower Bound on the Total Number of Workers) *In any (cost-) optimal solution to Problem (5.2),* $f + b + x \geq X_{total}$.

Proof: [By Contradiction] Assume we have a solution to Problem (5.2) where $f = f', b = b', x = x'$ and $f' + b' + x' < X_{total}$ (f', b', x' are non-negative integers). In particular, assume that $f' + b' + x' = X_{total} - 1$. Thus, in this solution, there are x' cross-trained workers that are switched between the back room and the front room according to some switching policy $< K, b^* >$. This solution can be transformed into an identical solution (in terms of the number of workers) as follows. Assume we have $X_{total} - 1$ cross-trained workers: constrain f' of those workers to always be in the front room, constrain $b' + b^*$ of those workers to always be in the back room, and switch $x' - b^*$ of those workers according to policy K. We now have a solution to (5.2) in which $x = X_{total} - 1$, $f = 0$ and $b = 0$. Recall that $x = X_{total}, f = 0, b = 0$ is the lowest-cost cross-trained-only solution. Since the cost of a cross-trained worker is greater than 0, we have a contradiction: our new solution has lower cost than that the lowest-cost cross-trained-only solution. Therefore, the optimal solution to problem (5.2) without specialized workers, $x = X_{total}, f = 0, b = 0$ provides a lower bound on the total number of workers required. ■

Three of the cost cases are unaffected by the change in policy but two are. Below, we discuss the implications of the policy change on each of the cost cases in turn.

5.5.1 Cost Cases Affected by the Policy Change

Given the $< K, b^* >$ policy definition, it is always possible to replace some number of back room workers in a feasible configuration by the same number of cross-trained workers and retain this configuration's feasibility. This results in the following claims for Cost Cases 1 and 4. In both cases, the cost of the optimal solution will be smaller than, or equal to, the cost of the optimal solution with the K-policy formulation. This is true because $c_x \leq c_b$ and the

number of cross-trained workers will, consequently, be higher in the optimal solution under the $< K, b^* >$-policy definition than under the K-policy definition.

5.5.1.1 Cost Case 1: $c_b > c_x > c_f$

Proposition 5.5.1 *If $c_b > c_x > c_f$, then the optimal solution is always of the form $x = B_{total}, f = f', b = 0$, where $f' \leq F_{total}$ and f' is the smallest integer that makes this solution feasible.*

Proof: We start with solution $x = 0, f = F_{total}, b = B_{total}$, which is feasible by definition, and try to construct a smaller-cost feasible solution. Since $c_b > c_x$, substituting each of the B_{total} back room workers by cross-trained ones will result in a lower-cost (feasible) solution. Since some of the B_{total} cross-trained workers may help to satisfy the front room constraint, the cost of the resulting solution can be further decreased by reducing the number of front room workers. We therefore reduce f until the smallest possible value, f', that makes the solution $x = B_{total}, f = f', b = 0$ feasible for both constraints. It is not possible to modify this solution further in order to lower its cost and still retain its feasibility: x cannot be decreased because of the constraint $b + x \geq B_{total}$, f cannot be decreased because f' is the smallest value for f that makes this solution feasible, and x cannot be increased to replace some specialized front room workers because $c_x > c_f$.

Thus, the solution $x = B_{total}, f = f', b = 0$ is always optimal under the given cost assumptions. ■

This problem can be solved quite easily by first constraining x to equal B_{total}, b to equal 0, and then decreasing the value of f, starting from F_{total}, until it is not possible to find a feasible policy for the resulting combination of workers. The last value of f for which a feasible policy exists will be the required optimal value, f'.

It can be seen that, in one way, the problem is easier to solve under these cost assumptions when the new policy formulation is used. However, the search space of each sub-problem is

now larger because policies with various values of b^* have to be examined.

5.5.1.2 Cost Case 4: $c_x \leq c_f$ and $c_x \leq c_b$

Proposition 5.5.2 *If $c_x \leq c_f$ and $c_x \leq c_b$, then $x = X_{total}, f = 0, b = 0$ is always optimal.*

Proof: Suppose there exists a solution with better cost than the cost of solution $x = X_{total}, f = 0, b = 0$. This lower-cost solution will have the form $x = X_{total} - x', f = f', b = b'$. Since $x + f + b \geq X_{total}$, $X_{total} - x' + f' + b' \geq X_{total}$, which implies that, in any feasible solution, $f' + b' \geq x'$. The cost of the new solution is

$$c_x(X_{total} - x') + c_f(f') + c_b(b')$$

$$= c_x(X_{total}) - c_x(x') + c_f(f') + c_b(b')$$

$$\geq c_x(X_{total})$$

because $c_f(f') + c_b(b') - c_x(x') \geq c_f(f') + c_b(b') - c_x(f' + b') = f'(c_f - c_x) + b'(c_b - c_x)$, which is ≥ 0 due to the cost assumptions. (When equality holds, it is implied that there may be other solutions with the same cost.)

Thus, $x = X_{total}, f = 0, b = 0$ is optimal for these cost assumptions. ■

With assumptions $c_x \leq c_f$ and $c_x \leq c_b$, the problem can therefore be solved by finding the value of X_{total} (in other words, by solving Berman et al.'s problem P_2).

5.5.2 Cost Cases Unaffected by the Policy Change

Three of the cost cases are unaffected by the policy change: $c_f > c_x > c_b$ (Cost Case 2), $c_x \geq c_f + c_b$ (Cost Case 3), and $c_x \leq c_b + c_f$, $c_x \geq c_f$, $c_x \geq c_b$ (Cost Case 5). This happens because in all of them, $c_x > c_b$ or $c_x \geq c_b$ is true, and so permanently assigning cross-trained workers to the back room will never decrease the cost of a configuration. As a result, b^* is

always 0 in the optimal solution, and the problem reduces to finding a feasible policy K. An important consequence of this observation is that the method proposed in Chapter 4 remains valid with the more realistic policy definition.

5.6 Conclusions

In this chapter, we discuss the fact that the policy formulation of Berman et al. is not able to represent a set of solutions that appear reasonable in practice. An alternative policy formulation is presented which takes these solutions into account. It is shown that the cost assumptions for the problem of finding the optimal staffing configuration in a retail facility with back room and front room operations can be broken up into five categories. For each cost case, we prove that the optimal solution will be of a particular form and give an idea of how this optimal solution may be found. The form of the optimal solution in two of the five cost cases is directly affected by the policy definition used, while for the remaining three cases, the problem is always reduced to finding a feasible policy as it was originally defined in Chapter 4. Since Cost Case 5 is one of the cases unaffected by the policy definition, all of the material presented in Chapter 4 remains valid for the more realistic policy definition.

Chapter 6

Conclusions and Future Work

In this final chapter, we summarize the work presented in previous chapters, re-state the major contributions of this dissertation and present some possible directions for future work.

6.1 Summary and Contributions

6.1.1 Investigation of a Queueing Control Problem

In Chapter 3, we examined a queueing control problem which concerns a retail facility employing cross-trained workers. The retail facility has a front room, where customers arrive according to a stochastic process, and a back room, where the amount of work is known in advance. The goal of the problem is to determine a policy that specifies how to switch workers between the two rooms so as to minimize expected customer waiting time while ensuring that the expected number of workers in the back room is sufficient to complete all required tasks. In previous work, this problem has only been solved heuristically.

6.1.1.1 Complete Methods for a Problem Previously Solved Only Heuristically

This dissertation presented several complete methods for solving the optimal switching policy problem, all of which are based on constraint programming. The best pure constraint program-

ming model is based on closed-form expressions of the probabilities that are necessary for the calculation of the expected waiting time and the expected number of workers in the back room. When combined with shaving, an inference method from the CP literature, this model was able to prove optimality in 79.3% of instances that we experimented with. It also found the best-known solution in 97.6% of all these instances. The proposed hybrid approach, based on a combination of the best constraint programming model and the heuristic presented in the work of Berman et al. [15], was shown to be just as effective in proving optimality as the best pure CP method and to find good-quality solutions in a substantially shorter amount of time than any of the pure CP methods.

6.1.1.2 Constraint Programming for a Queueing Control Problem

To our knowledge, no previous work exists which attempts to apply constraint programming to queueing control optimization problems. In Chapter 3, we, firstly, proposed three constraint programming models for our queueing control problem of interest. By doing so, we demonstrated that CP modelling techniques, such as the use of dual variables and redundant constraints, are suitable for modelling queueing control optimization problems. Secondly, we applied two specialized CP inference methods to our problem, one of which was shown to be particularly effective. Thirdly, we experimentally demonstrated that our CP-based methods are able to find the optimal solution and prove its optimality in many problem instances within a reasonable amount of time. Implementation of these methods used only the predefined constraints available in standard CP solvers. Overall, Chapter 3 illustrated that constraint programming may be used to model and solve optimization problems arising in the field of queue control.

6.1.2 Investigation of a Queueing Design and Control Problem

6.1.2.1 Extension of an Existing Problem and Its Analysis

In Chapter 4, we presented a realistic extension of an existing queueing design and control problem. In particular, we considered a retail facility that can hire specialized, in addition to cross-trained, workers. The goal of the problem is to determine the number of cross-trained and specialized workers that should be hired so that the total staffing costs are minimized yet service level constraints in both rooms are satisfied.

We provided, in Chapter 5, an analysis of the structure of the optimal solution to this problem under different assumptions regarding the staffing costs of specialized and cross-trained workers. More specifically, we proved that, in four of the five possible cost cases, the optimal solution has a special form and can be found easily, while in order to solve instances in the fifth case, a more sophisticated approach is necessary. Additionally, as part of this analysis, we identified a weakness in the policy formulation proposed in previous work and suggested a new policy formulation. We included a discussion of the implications of the policy definition on the structure of the optimal solution to the problem.

6.1.2.2 Constraint Programming for a Queueing Design and Control Problem

We presented a method which uses logic-based Benders' decomposition and constraint programming for solving the optimal staff mix queueing design and control problem. The master problem is a queueing design problem of identifying the lowest-cost staffing configuration, and the sub-problem is a queueing control problem of finding a feasible control policy. In our logic-based Benders' decomposition approach, both are modelled and solved using constraint programming. In our experiments, this method performed well over a wide range of problem parameters, being able to find the optimal solution for most instances within a reasonable runtime. Therefore, we demonstrated that constraint programming can be used to solve a problem which deals with both optimal queue design and optimal queue control.

6.1.3 Integration of Constraint Programming and Queueing Theory

By showing that constraint programming can be a useful approach for queueing design and control problems, we created a link between existing methodologies of constraint programming and queueing theory. We also helped to broaden the scope of problems that can be solved by constraint programming as well as extended the range of methods available for solving queueing-based problems. More generally, we contributed to the recent work on the integration of constraint programming and operations research techniques.

6.2 Future Work

Possible directions for future work include exploring extensions of the problems addressed in the preceding chapters, investigating applications of constraint programming to other queueing design and control problems from the literature, as well as addressing more general problems which may be solved using combinations of methods from constraint programming and queueing theory.

6.2.1 Further Work on Problems of Chapters 3 and 4

The first direction for further work is to continue the examination of our two problems of interest. Such work may focus either on investigation of ideas that may help remedy the drawbacks of our methods or on direct extensions of our problems.

6.2.1.1 Possible Improvements

In Chapter 3, we presented three constraint programming models for the problem of finding the optimal times to switch cross-trained workers between two tasks (problem P_1). However, none of our methods were able to prove optimality in all our test instances. Thus, it would be interesting to see whether these methods could be improved. In particular, in Section 3.6.1, we observed that the reason for the lack of back-propagation in the *PSums* model may be the

complexity of the expression for L, or, more specifically, the fact that this expression contains decision variables in exponents. We could therefore consider creating a model based on the logarithms of probabilities rather than the probabilities themselves. Such a model may allow for more back-propagation and result in search being able to find better solutions or prove that none exist faster.

One may also attempt to reformulate the expressions for the $\beta Sum(k_i)$ variables in the *Dual* model in terms of the w_j variables. If this is possible, one could consider removing the k_i variables from the *Dual* model, which would require a substitute for the constraint $P(k_0) \sum_{i=0}^{N} \beta Sum(k_i) = 1$ to be derived. Since it appears that the reason for the relatively poor performance of the *Dual* is the large number of variables that have to be assigned at each step of shaving, removing the k_i variables in an effective manner may lead to a better model. We could also evaluate the effect of the redundant channelling constraint in the *Dual* on this model's effectiveness.

Furthermore, one could experiment with various variable and value ordering heuristics. As noted in the work of Smith [82], these may have a significant impact on the run-times of search. A detailed examination of shaving procedures based on addition of constraints on the values of two switching points as well as of possible combinations of these procedures with the ones discussed in Section 3.3 could also be interesting.

Any improvements of the methods used to solve problem P_1 resulting from the investigation of these ideas would translate almost directly to the method used for solving the sub-problem of the problem of finding the optimal staff mix (discussed in Chapter 4). In order to try to improve the performance of the overall Benders' decomposition method for that problem, one could attempt to derive stronger cuts (which would remove more infeasible staff configurations from consideration in the master problem and therefore would decrease the overall number of iterations) or to include some sort of a relaxation of the sub-problem in the master problem (which would allow the master problem to find solutions that are more likely to lead to feasible policies in the sub-problem).

6.2.1.2 Extensions

Future work may also focus on a variety of extensions of both the problem P_1 and the optimal staff mix problem.

Variable Arrival or Service Rates One way to extend problem P_1 is to consider the possibility of controlling parameters of the queue which are currently assumed to be fixed, such as the mean service rate, μ, or the mean arrival rate, λ.[1]

Firstly, suppose that μ depends on the amount of training the workers receive and that the training cost incurred by the facility per unit increase in μ can be estimated. We may then be interested in finding the value of μ which minimizes the expected training costs of the facility subject to constraints requiring a policy to exist which ensures that all work in the back room is performed and the customer waiting time is bounded from above.

Secondly, we can assume that we have some control over the arrival process. For example, customers may have to pay money for service or in order to enter the facility. In this case, setting the fee that the customer should be charged for service/entrance is equivalent to controlling the arrival rate into the system. Given a function which relates the arrival rate and the fee charged, it would be interesting to determine the optimal amount that the customers have to pay so that the facility's profit is maximized, and the service level constraints in both the back room and the front room are met.

The structure of both of these problems appears to be well-suited to a Benders' decomposition approach, such as the one proposed in Chapter 4: the master problem could determine the service rate, μ, or the arrival rate, λ, while the sub-problem could determine whether a satisfying switching policy exists with N cross-trained workers having service rate μ, and assuming an average arrival rate of λ.

We may, similarly, consider an extension of the optimal staff mix problem in which we have control over the service rates of specialized and cross-trained workers or over the arrival

[1]The idea of considering variable arrival and/or service rates was initially suggested by Dr. A. Azaron.

rate. Other extensions involving the control of service or arrival rates may arise if our general problem of switching servers between two job types is placed in another context (e.g. similar problems can be found in computer science applications, as shown by the work of Palmer & Mitrani [63] for example).

Switching Costs In problem P_1, it is assumed that switches of workers between the two rooms occur instantaneously and do not result in additional costs for the facility. In reality, however, switches may take time (e.g. workers may require time to re-adjust to the new type of work) or may incur cost (e.g. customers may be lost while a switch is occurring), depending on the particular environment in which the problem arises. This aspect of the problem is important, since, for example, frequent switches that take a long time may imply that the waiting time of customers is greater than estimated, or that significant costs are incurred due to lost customers. The problems considered in this dissertation could therefore be extended by the addition of switching cost (or switching time).

One way to incorporate this aspect into the problems is to determine the frequency of switches from the steady-state probabilities [13]. Recall that a switching policy $(k_0, k_1, \ldots, k_N)$ is defined in terms of switching points k_i, $i = 0, ..., N$, so that there are i workers in the front room whenever there are between $k_{i-1} + 1$ and k_i customers in the front room. Given this definition, it can be seen that switches to the front room occur when there are k_i customers in the front room, and an additional customer arrives. Similarly, switches to the back room occur when there are $k_i + 1$ customers, and a customer gets served and leaves. Therefore, the expected frequency of switches is the product of the long-run proportion of time when the system is in a state when a switch is possible, and the rate at which arrivals (or departures) occur when the system is in such a state. Let f_{front} represent the expected frequency of switches to the front room and f_{back} represent the expected frequency of switches to the back room. These quantities can then be expressed as

$$f_{front} = \sum_{i=0}^{N-1} \lambda P(k_i) = \lambda \sum_{i=0}^{N-1} P(k_i)$$

and

$$f_{back} = \sum_{i=0}^{N-1}(i+1)\mu P(k_i+1) = \mu \sum_{i=0}^{N-1}(i+1)P(k_i+1).$$

In the long-run, the expected frequency of switches to the back room and the expected frequency of switches to the front room should be equal. Thus, we will further denote the expected frequency by f and assume that it is calculated using the formula for f_{front} (using the formula for f_{back} should yield equivalent results).

Given this calculation of expected switching frequency, a constraint may be added to the problem requiring it to be smaller than some upper bound, the value of which will depend on the particular environment being modelled. Alternatively, one may be able to estimate the cost incurred per switch (either monetary or in terms of time), say t, and place a constraint on the expected cost of switching, $f \times t$, or incorporate this expected cost into the objective function. It should not be difficult to incorporate switching costs into our constraint programming models since only a new constraint or a new term in the objective function would be added. If a new constraint is added, it would have to be temporarily removed prior to the use of shaving procedures so as to ensure that these procedures would not make any incorrect inferences. Other modifications to the shaving algorithms may also be necessary.

The problem of finding the optimal number of cross-trained workers (Berman et al.'s P_2) and the problem of finding the optimal staff mix could also be extended by the incorporation of switching costs. In the optimal staff configuration problem, this could lead to more instances having the specialized-only solution as optimal, since specialized workers cannot incur any switching cost.

Minimization of Blocking Probability The system being modelled in problem P_1 has a fixed capacity, S, which means that customers who arrive when there are already S customers in the front room leave without getting served. The number of customers turned away may, in fact, be an even more important measure of the service quality of the facility than expected customer waiting time, since, if a large number of customers is turned away, the reputation

of the facility will drop, and it will be much less successful.[2] Thus, it may be interesting to investigate the problem of minimizing the probability of blocking subject to back room and front room constraints. Alternatively, a constraint could be included in the problem bounding this probability. Such a constraint could also be added to the sub-problem of the optimal staff mix problem.

Other Policies Clearly, the formulation of problem P_1 is based on a particular policy definition. Other policy types, such as one that is defined by two vectors, one stating when to switch workers to the front room and one stating when to switch them to the back room, are possible. An important direction for future work is to determine, theoretically, the optimal policy type for this problem [15] and then to create a model for optimizing the parameters of this policy. Of particular interest is the question of whether constraint programming would be able to solve the problem given a different (optimal or sub-optimal) policy type.

Considering a different policy formulation in the optimal staff configuration problem would change the structure of the sub-problem and, possibly, the way in which the sub-problem would be solved. However, the idea of Benders' decomposition for solving the overall problem would still be applicable. A different policy formulation could have an effect on the optimal staff mix, however, since the existence of a feasible policy in one particular policy class does not guarantee the existence of a policy respecting both back room and front room constraints in a different one.

Stochastic Back Room Work Another direct extension of problem P_1 is to suppose that back room work is also generated by a stochastic process.[3] We could then consider minimizing the expected waiting time of customers in the front room subject to a constraint ensuring that the expected "waiting time" of requests or the expected time of completion of tasks in the back room is bounded from above. Alternatively, one may want to minimize the latter quantity while

[2] These ideas have been suggested by Dr. S. A. Tarim.
[3] This idea has been suggested by Dr. K. N. Brown.

satisfying a constraint on the expected customer waiting time in the front room. Note that the resulting problem would be similar to the problem considered by Berman & Sapna-Isotupa [14], where the amount of work in the back room is assumed to be correlated with the amount of work in the front room.

In the case of the staff mix problem, stochastic back room work could be taken into account in the sub-problem by replacing the back room constraint with a constraint requiring expected waiting time of back room requests to be bounded from above.

Different Service Rates One of the main assumptions in the problem of finding the optimal staff mix is that all workers are equally skilled in performing work in both rooms. In reality, such an assumption may not hold, and specialized front room workers may have a higher rate of service in the front room than cross-trained workers. Thus, it is interesting to consider the same problem of finding the optimal staff mix, but with workers of different productivity. The simplest way to incorporate this aspect into the model is to assume that specialized front room workers have service rate μ_f, and cross-trained workers have service rate μ_x. The balance equations of the sub-problem would then have to be appropriately reformulated. Also, such assumptions imply that policy $\hat{K}$ directly depends on the number of specialized and cross-trained workers, not just the total number of workers in the front room as implied in Section 5.4.2. As a result, many of the proofs stated in Chapter 5 would require significant modifications.

One could also assume that each cross-trained worker has a different service rate, which would result in extensions to both the switching policy problem and the optimal staff mix problem. However, the resulting problem could become very complicated since one would have to know, each time a switch occurs, what the service rate of the switched worker is.

Other Extensions More complex problems could be created by combining the extensions proposed above. For example, we may consider incorporating both switching costs and variable service rates, or both the idea of stochastic back room work and switching costs, into either of our problems. It would be interesting to investigate whether such problems are practical and

whether constraint programming could be used to solve them.

6.2.2 Constraint Programming for Other Queueing Design and Control Problems

A related direction for future work is to determine the extent of the applicability of constraint programming to other queueing design and control problems. This direction can be explored by trying to apply constraint programming to the fundamental problems in queueing design and control[4] as well as to problems which are more complex than the ones considered in this dissertation.

6.2.2.1 Fundamental Problems

In Section 2.2, some of the fundamental queueing design and control literature has been discussed. Many of these papers give rise to numerical optimization questions which one may attempt to solve using constraint programming.

For example, it would be interesting to examine the applicability of CP to the design problems discussed by Hillier [46], in which either the optimal number of servers, the optimal service rate or the optimal arrival rate has to be determined. One concern with these problems is the small number of decision variables, which may make the use of CP to solve them unreasonable. However, an analysis of these problems may still prove useful because this would allow us to identify general patterns in the structure of queueing design models. These patterns could then be used for the creation of a global constraint that could be helpful in more complex problems involving queueing. For example, both the problems presented in this dissertation and the problems addressed by Hillier [46] contain expressions which are sums over a vector of variables in which the number of terms is also a variable (e.g. Equation (2.8)). These are hard to implement using pre-defined constraints of CP solvers, and so creating a global con-

[4]This direction has been suggested by Dr. J. Bookbinder.

straint together with an effective propagation algorithm may be worthwhile, especially if many queueing-based problems contain such expressions.

A similar situation arises with fundamental queueing control problems. Frequently, policies are defined by either one or two parameters [44, 60, 70], making resulting numerical optimization questions rather trivial. However, these problems may also serve as the starting point for the development of global constraints. An interesting fundamental queueing control paper which discusses a policy type defined by more than two parameters is that of Yadin & Naor [102]. In fact, it is stated that numerical optimization of its parameters is difficult, and no method for doing so is suggested. One may therefore attempt to find out whether constraint programming could be useful for solving this problem.

6.2.2.2 More Complex Problems

A complementary direction is to examine problems which have a more complicated structure than the ones addressed in this dissertation.

A starting point in this direction could be the work of Berman & Larson [13], who study the problem of switching workers between two rooms in a retail facility where the customers in the front room are divided into two categories, those “shopping” in the store and those at the checkout. Clearly, when there are many customers shopping in the store, it can be expected that the number of customers in the checkout queues will soon increase. Thus, the policy of assigning workers to serve customers at the checkout should depend on the total number of customers in the store (both the shoppers and the people at check-out). A policy therefore determines the number of workers in the front room depending on the number of customers in the checkout queue, the number of customers in the store but not at checkout, and the amount of work in the back room. In addition, switching time is explicitly taken into account. It would be interesting to investigate whether constraint programming can be successfully applied to this problem, especially since it is solved only heuristically by Berman & Sapna-Isotupa [13]. However, the balance equations are more complex in this problem, and it may be hard to

formulate them in a constraint programming model in a way that would ensure that unique values are assigned to the probability variables.

Bookbinder [17] addresses another practical problem based on queueing theory and involving the concept of switching.[5] The problem concerns the scheduling of airplane landings and take-offs on a single runway, which can be viewed as a queueing problem in which the runway "serves" two types of customers. Due to safety regulations, the time between a take-off and a landing is usually greater than the time between two take-offs or two landings. It is therefore necessary to find the optimal times to switch from serving landings to serving take-offs, and vice versa, so as to minimize the expected waiting time in both queues. This paper does not address the exact problem of finding an optimal policy, but it does present the balance equations necessary for constructing an optimization problem. Due to the similarity of this problem to the problems studied in this dissertation, it is possible that it could be effectively solved by constraint programming.

Elements of queueing design and control problems also arise in application areas such as computers and communication systems [1, 63, 64], storage area network and recovery systems [52], health care [39] and other areas [42, 87]. It could be interesting to address design and control problems from these application areas using CP.

Another extension of our work would be to apply constraint programming to optimization problems that are based on queueing networks. A queueing network consists of nodes which represent "service stations", which may be composed of one or several servers and have a buffer (queue) of either a finite or infinite size. A customer (or a job) usually has to be served at several service stations in sequence. Similarly to the queueing design and control problems discussed in the literature review chapter, the controllable parameters of a queueing network may be the arrival rate of jobs (into the network or to each service station), the service rates of servers at different nodes, the number of servers, and others [86]. The objective may be, for example, to minimize the time spent by each job in the system. Some problems dealing with

[5] The similarity between this problem and problem P_1 has been suggested by Dr. J. Bookbinder.

the control of service rates and/or arrival rates in queueing networks have been addressed by Azaron & Ghomi [2] and Azaron et al. [3]. Examination of these problems as well as of others from the area of queueing networks may result in the development of further links between queueing theory and constraint programming.

6.2.3 Further Integration of Queueing Theory and Constraint Programming

It would be interesting to investigate the applicability of CP to problems which are not strictly queueing design and control problems, but in which queueing theory is used for modelling part of a more complex structure. Such problems may arise in scheduling, for example. These usually involve scheduling jobs on a set of machines, and queueing theory may play a role if we assume that some jobs arrive according to a stochastic process, or job durations are uncertain. One practical problem with such characteristics is that of scheduling operating rooms. More specifically, operating rooms have to deal with both emergency cases and pre-scheduled surgeries. Minimizing expected waiting time of patients and maximizing utilization are two very important optimization questions arising in this context. In fact, given the vast amount of literature that applies queueing theory to problems from the health care sector [66], it may be possible to find other interesting problems based on queueing theory for which constraint programming may be useful. Complex problems based on queueing theory also arise in call centers [53], communication networks [68] as well as computer science applications [88, 89].

6.3 Conclusions

The central thesis of this dissertation is that constraint programming can be effective in solving queueing design and control optimization problems. We demonstrated this thesis by providing constraint programming-based methods for a queueing control problem from the literature and for an extension of this problem which deals with both queue design and control. We showed

that these methods perform well on a variety of problem instances. To our knowledge, this is the first work that has attempted to integrate the methodologies of constraint programming and queueing theory. In future work, the synthesis of these two distinct research areas may provide a framework for solving complex problems that could not be effectively addressed by either of these fields separately.

List of Symbols

Bibliography

[1] A.S. Alfa and I. Frigui. Discrete NT-policy single server queue with Markovian arrival process and phase type service. *European Journal of Operations Research*, 88:599–613, 1996.

[2] A. Azaron and S. M. T. F. Ghomi. Optimal control of service rates and arrivals in Jackson networks. *European Journal of Operational Research*, 147:17–31, 2003.

[3] A. Azaron, H. Katagiri, K. Kato, and M. Sakawa. Modelling complex assemblies as a queueing network for lead time control. *European Journal of Operational Research*, 174:150–168, 2006.

[4] F. Bacchus and P. van Beek. On the conversion between non-binary and binary constraint satisfaction problems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI'98)*, pages 311–318, 1998.

[5] T. Balafoutis and K. Stergiou. Algorithms for stochastic CSPs. In *Proceedings of the Twelfth International Conference on Principles and Practice of Constraint Programming (CP'2006)*, pages 44–58. Springer, 2006.

[6] P. Baptiste, P. Laborie, C. Le Pape, and W. Nuijten. Constraint-based scheduling and planning. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 22, pages 761–799. Elsevier, 2006.

[7] R. Barták. Effective modeling with constraints. In *Proceedings of the Fifteenth International Conference on Applications of Declarative Programming and Knowledge Management*, pages 149–165, 2004.

[8] R. Barták. Constraint propagation and backtracking-based search: A brief introduction to mainstream techniques of constraint satisfaction. Lecture Notes for the First International Summer School on Constraint Programming, 2005. Available at: http://www.math.unipd.it/˜frossi/cp-school/.

[9] J. C. Beck, A. J. Davenport, E. D. Davis, and M. S. Fox. The ODO project: Toward a unified basis for constraint-directed scheduling. *Journal of Scheduling*, 1(2):89–125, 1998.

[10] J. C. Beck and S. Prestwich. Exploiting dominance in three symmetric problems. *Fourth International Workshop on Symmetry and Constraint Satisfaction Problems*, 2004.

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

[12] T. Benoist, E. Bourreau, Y. Caseau, and B. Rottembourg. Towards stochastic constraint programming: A study of on-line multi-choice knapsack with deadlines. In *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming*, pages 61–76, 2001.

[13] O. Berman and R. Larson. A queueing control model for retail services having back room operations and cross-trained workers. *Computers and Operations Research*, 31(2):201–222, 2004.

[14] O. Berman and K.P. Sapna-Isotupa. Optimal control of servers in front and back rooms with correlated work. *IIE Transactions*, 37(2):167–173, 2005.

[15] O. Berman, J. Wang, and K. P. Sapna. Optimal management of cross-trained workers in services with negligible switching costs. *European Journal of Operations Research*, 167(2):349–369, 2005.

[16] C. Bessiere. Constraint propagation. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 3, pages 29–83. Elsevier, 2006.

[17] J. Bookbinder. Multiple queues of aircraft under time-dependent conditions. *INFOR: Canadian Journal of Operational Research and Information Processing*, 24(4):280–288, 1986.

[18] L. Bordeaux and H. Samulowitz. On the stochastic constraint satisfaction framework. In *Proceedings of the ACM Symposium on Applied Computing (SAC'07)*, 2007.

[19] K. N. Brown and I. Miguel. Uncertainty and change. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 21, pages 731–760. Elsevier, 2006.

[20] M. J. Brusco and T. R. Johns. Staffing a multiskilled workforce with varying levels of productivity: An analysis of cross-training policies. *Decision Sciences*, 29(2):499–515, 1998.

[21] Y. Caseau and F. Laburthe. Cumulative scheduling with task intervals. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 363–377. MIT Press, 1996.

[22] T. Cezik and P. L'Ecuyer. Staffing multiskill call centers via linear programming and simulation. *Management Science*, To Appear in 2007. Available from http://www.iro.umontreal.ca/˜lecuyer/ myftp/papers/sbrms.pdf.

[23] P. Chevalier and N. Tabordon. Overflow analysis and cross-trained servers. *International Journal of Production Economics*, 85:47–60, 2003.

[24] Wikipedia contributors. Golomb ruler. Wikipedia, The Free Encyclopedia, 2007. Available at: http://en.wikipedia.org/wiki/Golomb_ruler.

[25] T. Crabill, D. Gross, and M. Magazine. A classified bibliography of research on optimal design and control of queues. *Operations Research*, 25(2):219–232, 1977.

[26] R. Dechter and A. Dechter. Belief maintenance in dynamic constraint networks. In *Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI'88)*, pages 37–42. AAAI Press/MIT Press, 1988.

[27] S. Demassey, C. Artigues, and P. Michelon. Constraint-propagation-based cutting planes: An application to the resource-constrained project scheduling problem. *INFORMS Journal on Computing*, 17(1):52–65, 2005.

[28] D. Dubois, H. Fargier, and H. Prade. Possibility theory in constraint satisfaction problems. *Applied Intelligence*, 6:287–309, 1996.

[29] X. Fan-Orzechowski and E. Feinberg. Optimal admission control for a markovian queue under the quality of service constraint. In *Proceedings of the Forty Fourth IEEE Conference on Decision and Control, and the European Control Conference*, pages 1729–1734, 2005.

[30] H. Fargier and J. Lang. Uncertainty in constraint satisfaction problems: a probalistic approach. In *Proceedings of the Second European Conference on Symbolic and Qualitative Approaches to Reasoning with Uncertainty*, pages 97–104, 1996.

[31] H. Fargier, J. Lang, R. Martin-Clouaire, and T. Schiex. A constraint satisfaction framework for decision under uncertainty. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 167–174. Morgan Kaufmann, 1995.

[32] H. Fargier, J. Lang, and T. Schiex. Mixed constraint satisfaction: a framework for decision problems under incomplete knowledge. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 175–180, 1996.

[33] E. A. Feinberg and A. Shwartz, editors. *Handbook of Markov Decision Processes*. International Series in Operations Research and Management Science. Springer, first edition, 2005.

[34] D. W. Fowler and K. N. Brown. Branching constraint satisfaction problems for solutions robust under likely changes. In *Proceedings of the Sixth International Conference on Principles and Practice of Constraint Programming. CP'2000*, volume 1894, pages 500–504. Springer, 2000.

[35] D. W. Fowler and K. N. Brown. Branching constraint satisfaction problems and Markov decision problems compared. *Annals of Operations Research*, 118:85–110, 2003.

[36] M. S. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling*. PhD thesis, Carnegie Mellon University, Intelligent Systems Laboratory, The Robotics Institute, Pittsburgh, PA., 1983. CMU-RI-TR-85-7.

[37] E. C. Freuder and A. K. Mackworth. Constraint satisfaction: An emerging paradigm. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 2, pages 13–27. Elsevier, 2006.

[38] A. M. Frisch, I. Miguel, and T. Walsh. Extensions to proof planning for generating implied constraints. In *Proceedings of the Ninth Symposium on the Integration of Symbolic Computation and Mechanized Reasoning (Calculemus'01)*, pages 130–141, 2001.

[39] K. N. Gaur. Application of controlled queuing model in health care services. In *First International Conference on Operations and Quantitative Management*, volume 2, pages 630–637, 1997.

[40] R. F. Gebhard. A queueing process with bilevel hysteretic service-rate control. *Naval Research Logistics Quarterly*, 14:55–68, 1967.

[41] W. Grassmann, X. Chen, and B. R. K. Kashyap. Optimal service rates for the state-dependent M/G/1 queues in steady-state. *Operations Research Letters*, 29:57–63, 2001.

[42] D. Gross and C. Harris. *Fundamentals of Queueing Theory*. John Wiley & Sons, Inc., 1998.

[43] E. Hebrard, B. Hnich, and T. Walsh. Super solutions in constraint programming. In *Proceedings of First International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'04)*, volume 3011, pages 157–172. Springer, 2004.

[44] M. Hersh. Allocation processes with variable channel queues. *Management Science*, 20(2):203–213, 1973.

[45] D. P. Heyman. Optimal operating policies for M/G/1 queueing systems. *Operations Research*, 16:362–382, 1968.

[46] F. S. Hillier. Economic models for industrial waiting line problems. *Management Science*, 10(1):119–130, 1963.

[47] B. Hnich, J. Richardson, and P. Flener. Towards automatic generation and evaluation of implied constraints. Technical Report 2003-014, Department of Information Technology, Uppsala University, Sweden, 2003.

[48] B. Hnich, B. M. Smith, and T. Walsh. Dual modelling of permutation and injection problems. *Journal of Artificial Intelligence Research*, 21:357–391, 2004.

[49] J. N. Hooker and G. Ottosson. Logic-based Benders' decomposition. *Mathematical Programming*, 96:33–60, 2003.

[50] H. Hoos and E. Tsang. Local search methods. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 5, pages 135–167. Elsevier, 2006.

[51] O. Kella. Optimal control of the vacation scheme in an M/G/1 queue. *Operations Research*, 38:724–728, 1990.

[52] S.-K. Kim and J.H. Dshalalow. Stochastic disaster recovery systems with external resources. *Mathematical and Computer Modelling*, 36:1235–1257, 2002.

[53] G. Koole and A. Mandelbaum. Queueing models of call centers: an introduction. *Annals of Operations Research*, 113:41–59, 2002.

[54] V. Kumar. Algorithms for constraint satisfaction problems: A survey. *AI Magazine*, pages 32–44, 1992.

[55] S. Manandhar, S. A. Tarim, and T. Walsh. Scenario-based stochastic constraint programming. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'2003)*, 2003.

[56] M. Martin and J. R. Artalejo. Analysis of an M/G/1 queue with two types of impatient units. *Advances in Applied Probability*, 27:840–861, 1995.

[57] P. Martin and D. B. Shmoys. A new approach to computing optimal schedules for the job shop scheduling problem. In *Proceedings of the Fifth Conference on Integer Programming and Combinatorial Optimization*, pages 389–403, 1996.

[58] P. Meseguer, F. Rossi, and T. Schiex. Global constraints. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 9, pages 281–328. Elsevier, 2006.

[59] S. Minton, M. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[60] J. J. Moder and C. R. Phillips, Jr. Queueing with fixed and variable channels. *Operations Research*, 8:218–231, 1962.

[61] P. M. Morse. *Queues, Inventories and Maintenance*. John Wiley & Sons, Inc., 1958.

[62] R. Nobel and H. Tijms. Optimal control of a queueing system with heterogeneous servers and setup costs. *IEEE Transactions on Automatic Control*, 45(4):781–785, 2000.

[63] J. Palmer and I. Mitrani. Optimal server allocation in reconfigurable clusters with multiple job types. In *Proceedings of the Computational Science and Its Applications International Conference*, pages 76–86, 2004.

[64] W. L. Pearn, J.-C. Ke, and Y. C. Chang. Sensitivity analysis of the optimal management policy for a queuing system with a removable and non-reliable server. *Computers and Industrial Engineering*, 46:87–99, 2004.

[65] E. Pinker and R. Shumsky. The efficiency-quality trade-off of cross-trained workers. *Manufacturing and Service Operations Management*, 2(1):32–48, 2000.

[66] J. Preater. A bibliography of queues in health and medicine. Technical Report 01-1, Department of Mathematics, Keele University, 2001. Available at www.chcm.ubc.ca/comm682/QueueingBib.pdf.

[67] P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[68] G. Pujolle. Queueing systems for modelling ATM networks. *IFIP Transactions C (Communication Systems)*, C-5:301–322, 1992.

[69] J.-C. Régin. A filtering algorithm for constraints of difference in CSPs. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, volume 1, pages 362–367, 1994.

[70] J. Romani. Un modelo de la teoría de colas con número variable de canales. *Trabajos Estadistica*, 8:175–189, 1957.

[71] M. Rosenshine and R. C. Rue. Optimal control of entry of many classes of customers to an M/M/1 queue. *Naval Research Logistics Quarterly*, 28(3):489–495, 1981.

[72] F. Rossi, P. van Beek, and T. Walsh. Introduction. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 1, pages 3–12. Elsevier, 2006.

[73] S. Z. Selim. Time-dependent solution and optimal control of a bulk service queue. *Journal of Applied Probability*, 34:258–266, 1997.

[74] R. F. Serfozo and F. V. Lu. M/M/1 queueing decision processes with monotone hysteretic policies. *Operations Research*, 32:1116–1132, 1984.

[75] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher and J.-F. Puget, editors, *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming (CP'98)*, pages 417–431. Springer-Verlag, 1998.

[76] J. Slomp, J. A. C. Bokhorst, and E. Molleman. Cross-training in a cellular manufacturing environment. *Manufacturing and Service Operations Management*, 48:609–624, 2005.

[77] B. M. Smith. Reducing symmetry in a combinatorial design problem. In *Proceedings of CP-AI-OR'01, the International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 2001.

[78] B. M. Smith. A dual graph representation of a problem in 'Life'. In *Principles and Practice of Constraint Programming (CP'2002)*, pages 402–414, 2002.

[79] B. M. Smith. Constraint Programming in Practice: Scheduling a Rehearsal. Technical Report APES-67-2003, APES Research Group, September 2003. Available from http://www.dcs.st-and.ac.uk/˜apes/apesreports.html.

[80] B. M. Smith. Search Strategies for Optimization: Modelling the SONET Problem. Technical Report APES-69-2003, APES Research Group, August 2003. Available from http://www.dcs.st-and.ac.uk/˜apes/apesreports.html.

[81] B. M. Smith. Modelling for constraint programming. Lecture Notes for the First International Summer School on Constraint Programming, 2005. Available at: http://www.math.unipd.it/˜frossi/cp-school/.

[82] B. M. Smith. Modelling. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 11, pages 377–406. Elsevier, 2006.

[83] B. M. Smith, K. Stergiou, and T. Walsh. Using auxiliary variables and implied constraints to model non-binary problems. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'00)*, pages 182–187, 2000.

[84] S. Stidham, Jr. On the optimality of single-server queueing systems. *Operations Research*, 18(4):708–732, 1970.

[85] S. Stidham, Jr. Optimal control of admission to a queueing system. *IEEE Transactions on Automatic Control*, AC-30(8):705–713, 1985.

[86] S. Stidham, Jr. and R. Weber. A survey of Markov decision models for control of networks of queues. *Queueing Systems*, 13:291–314, 1993.

[87] L. Tadj and G. Choudhury. Optimal design and control of queues. *Sociedad de Estadìstica e Investigación Operativa, Top*, 13(2):359–412, 2005.

[88] H. Takagi. *Stochastic analysis of computer and communication systems*. North-Holland, 1990.

[89] H. Takagi. A bibliography of books on queueing analysis and performance evaluation. *Automatic Control and Computer Sciences*, 26(3):22–40, 1992.

[90] S. A. Tarim, S. Manandhar, and T. Walsh. Stochastic constraint programming: A scenario-based approach. *Constraints*, 11(1):53–80, 2006.

[91] S. A. Tarim and A. Miguel. A hybrid Benders' decomposition method for solving stochastic constraint programs with linear recourse. In *Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming*, pages 133–148, 2005.

[92] S. A. Tarim and B. M. Smith. Constraint programming for computing non-stationary (R, S) policy. *European Journal of Operations Research*, 2007. To Appear.

[93] D. Terekhov and J. C. Beck. Solving a stochastic queueing control problem with constraint programming. In *Proceedings of the Fourth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'07)*, pages 303–317. Springer-Verlag, 2007.

[94] D. Terekhov, J. C. Beck, and K. N. Brown. Solving a stochastic queueing design and control problem with constraint programming. In *Proceedings of Twenty-Second National Conference on Artificial Intelligence (AAAI'07)*, 2007.

[95] P. van Beek. Backtracking search algorithms. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 4, pages 85–134. Elsevier, 2006.

[96] M. R. C. van Dongen. Beyond singleton arc consistency. In *Proceedings of the Seventeenth European Conference on Artificial Intelligence (ECAI'06)*, pages 163–167, 2006.

[97] W.-J. van Hoeve. The alldifferent constraint: A survey. In *Proceedings of the Sixth Annual Workshop of the ERCIM Working Group on Constraints*, 2001.

[98] W.-J. van Hoeve and I. Katriel. Global constraints. In F. Rossi, P. van Beek, and T. Walsh, editors, *Handbook of Constraint Programming*, chapter 6, pages 169–208. Elsevier, 2006.

[99] G. Verfaillie and N. Jussien. Constraint solving in uncertain and dynamic environments: A survey. *Constraints*, 10(3):253–281, 2005.

[100] R. J. Wallace and E. C. Freuder. Stable solutions for dynamic constraint satisfaction problems. In *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming (CP'98)*, pages 447–461, 1998.

[101] T. Walsh. Stochastic constraint programming. In *Proceedings of the Fifteenth European Conference on Artificial Intelligence*, pages 111–115, 2002.

[102] M. Yadin and P. Naor. On queueing systems with variable service capacities. *Naval Research Logistics Quarterly*, 14:43–54, 1967.

[103] N. Yorke-Smith and C. Gervet. Certainty closure: A framework for reliable constraint reasoning with uncertainty. In *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP'2003)*, volume 2833, pages 769–783. Springer, 2003.