INFORMS Journal on Computing
Vol. 21, No. 4, Fall 2009, pp. 549–561
ISSN 1091-9856 | EISSN 1526-5528 | 09 | 2104 | 0549

DOI 10.1287/ijoc.1080.0307
© 2009 INFORMS

A Constraint Programming Approach for Solving a Queueing Design and Control Problem

Daria Terekhov, J. Christopher Beck
Department of Mechanical and Industrial Engineering, University of Toronto, Toronto, Ontario M5S 3G8, Canada
{dterekho@mie.utoronto.ca, jcb@mie.utoronto.ca}

Kenneth N. Brown
Cork Constraint Computation Centre, Department of Computer Science, University College Cork,
Cork, Ireland, k.brown@cs.ucc.ie

A facility with frontroom and backroom operations has the option of hiring specialized or cross-trained workers. Cross-trained workers can be switched between the two rooms depending on demand but are more expensive than specialized ones. Assuming stochastic customer arrival and service times, we seek a smallest-cost combination of cross-trained and specialized workers, together with a policy for switching the cross-trained workers between the rooms, which satisfies constraints on the expected customer waiting time and expected number of workers in the back room. A constraint programming approach using logic-based Benders' decomposition is presented. Experimental results demonstrate the strong performance of this approach across a wide variety of problem parameters. This paper provides one of the first links between queueing optimization problems and constraint programming.

Key words: constraint programming; queues; optimization; hybrid algorithms
History: Accepted by John Hooker, Area Editor for Constraint Programming and Optimization; received October 2007; revised August 2008; accepted September 2008. Published online in *Articles in Advance* February 5, 2009.

1. Introduction

Constraint programming (CP) has proven to be a successful technique for solving large-scale and complex deterministic resource allocation problems (Baptiste et al. 2006, Fox 1983). More recently, there has also been work in the artificial intelligence community on developing methods for stochastic versions of such problems (Brown and Miguel 2006). Resource allocation under uncertainty is also of interest in operations research (OR). In particular, queueing theory addresses problems related to the optimal design and control of queues and allows one to determine, for example, how to assign servers to different types of jobs or how many servers to employ so as to optimize some measure of system performance (Tadj and Choudhury 2005, Gross and Harris 1998). The integration of these two powerful techniques, constraint programming and queueing theory, offers the potential for solving more complex stochastic resource allocation problems.

In this work, we generalize an existing queueing design and control problem (Berman et al. 2005) and propose a complete hybrid solution technique combining logic-based Benders' decomposition (Hooker and Ottosson 2003), an extension of an existing heuristic, and CP. This paper has two main contributions. First, we provide a complete method for a generalization of a queueing design and control problem from the literature that has only been solved using a heuristic technique. Second, this paper creates a link between constraint programming and queueing theory: it establishes CP as a method for solving queueing optimization problems.

The paper is organized as follows. Section 2 presents a detailed description of our queueing design and control problem. In §3, we discuss related work. In §4, the details of our overall approach for solving the problem are presented. Experimental results and their analysis are given in §§5 and 6. Some ideas for further work are presented in §7. Section 8 concludes the paper.

2. Background

We consider a facility, such as a bank, with backroom and frontroom operations. In the front room, the workload depends on a stochastic process of customer arrivals and service times. If all frontroom workers are busy, customers form a queue and wait to be served. In the back room, tasks include sorting of material and paperwork, and do not directly depend on customer arrivals. The facility can hire either cross-trained workers, who are able to perform tasks in both rooms, or specialized workers, who are able only to serve customers or only to work in the back room. Cross-trained workers provide flexibility since they may be switched between the back room and the front room depending on demand. However, cross-trained workers are more expensive than specialized ones because they possess more skills. Managers of the facility are therefore interested in finding the optimal number of workers of each type and, if this combination includes cross-trained workers, in knowing when to switch them between the two rooms.

The current work addresses a generalization of two problems studied by Berman et al. (2005), who assume that *only* cross-trained workers can be hired by the facility. The objective of the Berman et al. (2005) problem P_1 is to determine when to switch workers between the two rooms so that expected customer waiting time is minimized, but the requirement on the minimum number of backroom workers is met. In problem P_2, the goal is to find the minimum number of cross-trained workers such that a switching policy exists to meet constraints on the expected customer waiting time and on the expected number of backroom workers. Berman et al. (2005) propose two heuristics, also called P_1 and P_2, for solving these problems.

2.1. Problem Description

Using the notation of Berman et al. (2005), let S denote the maximum number of customers allowed in the front room at any time. When there are S customers present, arriving customers are blocked from entering the facility. To ensure that all backroom work is completed, there is a known minimum requirement, B_l, for the expected number of workers in the back room. 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 cost are negligible. Customers arrive according to a Poisson process with rate λ. Customer service times follow an exponential distribution with rate μ.

We extend P_2 to allow specialized front and backroom workers in addition to cross-trained ones. Given a different staffing cost for each type of worker, the goal of this generalization is 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 backroom work is completed.

Let f be the number of specialized frontroom workers, b the number of specialized backroom 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 the worker is specialized or cross-trained. Similarly, cross-trained and specialized employees working in the back room are assumed to be able to perform backroom tasks equally well. Denote the staffing costs as c_f, c_b, and c_x, respectively, for frontroom, backroom, and cross-trained workers. We assume that $c_x \geq c_f > 0$, $c_x \geq c_b > 0$, and $c_x \leq c_f + c_b$. See the work of Terekhov and Beck (2009) for an analysis of the problem under other cost assumptions.

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. 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. Previously, no methods have been proposed for solving this problem, and even its special case, when there are only cross-trained workers in the facility, has only been addressed heuristically.

Formally, the problem of finding the optimal staff mix given frontroom and backroom constraints can be stated as

$$\begin{aligned} \text{minimize} \quad & c_f f + c_b b + c_x x \\ \text{s.t.} \quad & W_q \leq W_u, \\ & B \geq B_l. \end{aligned} \tag{1}$$

2.2. Switching Policy

An essential part of the problem is the question of how workers are to be switched between the two rooms. We assume that these decisions are made using a policy, that is, an a priori rule that specifies when a switch should be made. We extend the policy formulation proposed by Berman et al. (2005) to the case when specialized, in addition to cross-trained, workers can be hired. Thus, in this paper, a policy is defined 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. In other words, the number of workers in the front room is always equal to $\max(f, i)$. Switching point k_{x+f} is a constant and is equal to S. Thus, a policy is a vector $K = (k_0, k_1, \ldots, k_{x+f})$ with $k_i < k_{i+1}$ 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 one busy worker in the front room, and whenever the number of customers is 2 or 3, there are two 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 four customers in the room, there are three busy workers, remains valid, even though one of the busy workers is cross-trained.

It is important to realize that other policy types could be used for modelling switching decisions. For example, an alternative policy definition could be based on two sets of switching points, one for switching workers to the front room and one for switching workers to the back room, as in the work by Berman and Larson (2004). In this paper, however, we address the problem using only the policy form defined above. The quality of this policy type is discussed in a related paper (Terekhov and Beck 2009).

3. Related Work

The problem addressed in this paper belongs to the class of problems usually referred to as "optimal design and control of queues," which has applications in many areas such as communication systems and scheduling (Tadj and Choudhury 2005). Problems dealing with queue design are static in nature: the goal is to find optimal values for parameters that, once determined, become fixed characteristics of the queueing system, such as the maximum number of allowed customers or the total number of available servers (Gross and Harris 1998). Problems dealing with queue control are dynamic: the goal is to determine an optimal action to take when the queue is in a particular state (Gross and Harris 1998). The problem we address in this paper deals with both queue design and queue control, since we would like to determine how many workers of each type should be hired by the facility (a fixed characteristic of the system) and how cross-trained workers should be switched between tasks (what action should be taken in a particular state of the queue).

As noted earlier, our paper is closely linked with the work of Berman et al. (2005). In our previous work (Terekhov and Beck 2008), we proposed several constraint programming methods for solving the queue control problem of Berman et al. (2005) (problem P_1). However, to our knowledge, no papers have examined applying CP to a problem that involves both optimal queue design and control.

Related work also includes papers by Berman and Larson (2004), and Berman and Sapna-Isotupa (2005), both of which consider a retail facility with two rooms and cross-trained workers. Berman and Larson (2004) assume that there are two kinds of customers in the front room: ones who are "shopping" and ones who are at the checkout. They use heuristics to determine the minimum number of cross-trained workers that should be hired. Berman and Sapna-Isotupa (2005) consider the problem of minimizing the number of cross-trained workers in a facility where the amount of backroom work is correlated with the number of customers in the front room. They model the problem as a Markov decision problem and use a linear programming approach to solve it. One possible direction for further work is therefore to examine the applicability of our method to the case when there is correlation between frontroom and backroom tasks.

Our problem of interest deals with the management of cross-trained workers. Various problems involving this subject have been considered in the literature using simulation (Chevalier and Tabordon 2003) and linear and mixed-integer programming (Cezik and L'Ecuyer 2008, Batta et al. 2007, Brusco 2008). However, to our knowledge, CP has not been applied to such problems.

4. Logic-Based Benders' Decomposition Approach

Benders' decomposition was originally developed for solving mixed-integer programming problems (Benders 1962). 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 subproblem, which are solved in an alternating fashion until an optimality criterion is satisfied. Each time the subproblem is solved, a constraint that 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 subproblem may be modelled and solved using linear or integer programming (Benders 1962), or CP, as in logic-based Benders' decomposition (Hooker and Ottosson 2003, Tarim and Miguel 2005).

Given our previous work (Terekhov and Beck 2008), 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 subproblem of finding a switching policy that satisfies the backroom and frontroom service-level constraints given an employee configuration (a queueing control problem).

Our approach is to first derive a set of constraints on the values of x, f, and b by solving problem (1) with $x=0$. These constraints are then used to form the master problem, which is solved 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 subproblem solution can be found, then the master solution is optimal. Otherwise, the cut ($x > x' \vee f > f' \vee b > b'$), where $\vee$ represents a logical disjunction, is added to the master problem. The master problem and the subproblem are then re-solved to determine if a feasible policy exists for the new master problem solution.

4.1. 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 frontroom workers sufficient to satisfy the waiting-time constraint, and B_{total}, the smallest number of specialized backroom workers needed to satisfy the backroom constraint.

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

$$W_q = \frac{P_0 \sum_{i=1}^{i=S} (\lambda/\mu)^{i-1} i/D(i)}{\mu[1 - (\lambda/\mu)^S P_0 (1/D(S))]} - \frac{1}{\mu}, \tag{2}$$

where

$$D(i) = \prod_{j=1}^{i} d(j), \qquad d(i) = \begin{cases} i & \text{if } i \leq f, \\ f & \text{otherwise,} \end{cases} \tag{3}$$

and

$$P_0 = 1 + \sum_{i=1}^{i=S} \left(\frac{\lambda}{\mu}\right)^i \frac{1}{D(i)}, \tag{4}$$

until the constraint $W_q \leq W_u$ is satisfied for some value of f. We derived this equation from the expression for W_q, which appears in the work of Berman et al. (2005) (see Equation (12) below) using the fact that, when only specialized workers are considered, the front room is a queue with exponential interarrival and service times, f servers, and a capacity of S (an M/M/f/S queue) (Gross and Harris 1998).

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

By definition of F_{total} and B_{total}, there have to be at least F_{total} cross-trained and frontroom workers in the facility in order for the constraint on W_q to be satisfied, and at least B_{total} cross-trained and backroom workers in order for the backroom constraint to be satisfied. Thus, constraints $f + x \geq F_{\text{total}}$ and $b + x \geq B_{\text{total}}$ are valid for problem (1). In addition, in any feasible solution, the number of specialized frontroom workers does not ever have to exceed F_{total}, and the number of specialized backroom 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, $F_{\text{total}} - 1$ is an upper bound for f and $B_{\text{total}} - 1$ is an upper bound for b. $F_{\text{total}} + B_{\text{total}} - 1$ 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 to satisfy the frontroom constraint, and at least B_{total} workers in the facility to satisfy the backroom constraint, it is clear that at least $\max(F_{\text{total}}, B_{\text{total}})$ 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 $c_f F_{\text{total}}$ and $c_b B_{\text{total}}$ 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. Master Problem

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

$$\begin{aligned} \text{minimize} \quad & \text{cost} = c_f f + c_b b + c_x x \\ \text{s.t.} \quad & f + x \geq F_{\text{total}}, \\ & b + x \geq B_{\text{total}}, \\ & 0 \leq f \leq F_{\text{total}} - 1, \\ & 0 \leq b \leq B_{\text{total}} - 1, \\ & 1 \leq x \leq F_{\text{total}} + B_{\text{total}} - 1, \\ & f + b + x \geq \max(F_{\text{total}}, B_{\text{total}}), \\ & \max(c_f F_{\text{total}}, c_b B_{\text{total}}) \\ & \quad \leq \text{cost} \leq c_f F_{\text{total}} + c_b B_{\text{total}}, \\ & \mathit{cuts} \end{aligned} \tag{5}$$

where *cuts* are constraints that are added to the master problem each time the subproblem 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 subproblem. We solve the master problem with a CP model identical to (5). As it is a simple minimization problem with three integer variables, finding a solution is trivial.

4.3. Solving the Subproblem

Given values for f, b, and x, the goal of the subproblem is to find a policy K such that the constraints $W_q \leq W_u$ and $B \geq B_l$ are satisfied. The subproblem is very similar to problem P_1 of Berman et al. (2005).

Therefore, the method we use for solving the subproblem is a modification of the *PSums-P_1* Hybrid method proposed by Terekhov and Beck (2008), which combines a CP model with the Berman et al. heuristic. More specifically, to solve the subproblem, we run a modified version of the Berman et al. heuristic P_1 followed by a combination of shaving and search.

In this section, we review the definition of a switching policy and present the equivalents of the Berman et al. (2005) expressions for calculating the quantities of interest given that there may be specialized workers in the facility. These expressions are used throughout a modified version of the Berman et al. heuristic, which is presented in §4.3.2. We then present a CP model of the subproblem, which we refer to as *PSumsSubproblem*, and the details of shaving procedures used in conjunction with this model.

4.3.1. Switching Policy. Recall that a policy is 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. That is, it is a vector $K = (k_0, k_1, \ldots, k_{x+f})$, 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 $k_i \geq i$ $\forall\, i \geq f$.

This policy formulation can be seen as a special case of the policy formulation of Berman et al. (2005), with the first f switching points always being fixed to their minimum possible values. Therefore, by replacing each occurrence of N by $x+f$ in Theorem 1 of Berman et al. (2005), we can generalize this theorem to the case when specialized workers can be employed in the facility:

Theorem 1 (Berman et al. 2005). *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) $F(K) \leq F(K')$, *and* (c) $B(K) \geq B(K')$.

As a result of Theorem 1, we can define the equivalents of the Berman et al. policies $\widehat{K}$ and $\widehat{\widehat{K}}$. In particular, for the problem involving three types of workers, the policy yielding the greatest possible values of W_q and B is

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

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

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

The notions of types 1 and 2 components can be defined in a manner similar to Berman et al. (2005). 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 paper, 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.

4.3.2. Modified Berman et al. (2005) Heuristic. Given the above policy formulation, the subproblem can be modelled analogously to the way Berman et al. model problem P_1. In particular, to calculate the quantities of interest (e.g., W_q), a set of probabilities $P(j)$, for j between 0 and S, is 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

$$P(j)\lambda = P(j+1)i\mu,\\ j = k_{i-1}, k_{i-1}+1, \ldots, k_i - 1 \quad i = 1, \ldots, x+f. \tag{6}$$

Consequently, each probability can be calculated using the following expressions:

$$P(j) = \beta_j P(k_0), \tag{7}$$

where

$$\beta_j = \begin{cases} 1 & \text{if } j = k_0, \\ (\lambda/\mu)^{j-k_0}(1/i)^{j-k_{i-1}} X_i & \\ & \text{if } k_{i-1}+1 \leq j \leq k_i \quad i = 1, \ldots, x+f, \end{cases} \tag{8}$$

$$X_i = \prod_{g=1}^{i-1} \left(\frac{1}{g}\right)^{k_g - k_{g-1}} \qquad i = 2, \ldots, x+f,\ X_1 \equiv 1,$$

and

$$P(k_0) \sum_{j=0}^{S} \beta_j = 1. \tag{9}$$

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

$$F_{\text{cross}} = \sum_{i=f+1}^{x+f} \sum_{j=k_{i-1}+1}^{k_i} iP(j). \tag{10}$$

Because 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 $b + x - F_{\text{cross}}$.

The expected number of customers in the front room is exactly the same as in problem P_1:

$$L = \sum_{j=k_0}^{S} jP(j). \tag{11}$$

The expected waiting time in the queue can be obtained by replacing N by $x+f$ in the W_q expression of Berman et al. (2005):

$$W_q = \frac{L}{\lambda(1-P(k_{x+f}))} - \frac{1}{\mu}. \tag{12}$$

Based on the above equations, Theorem 1 and the definitions of $\widehat{K}$ and $\widehat{\widehat{K}}$, the equivalent of the Berman et al. P_1 heuristic for this problem, which will be further referred to as heuristic P_3, can be stated as

Step 1. Start with $K = \widehat{\widehat{K}}$.

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

Step 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 Step 5. Otherwise, set $k_{j^*} = k_{j^*} - 1$. If $B(K) < B_l$, set $J = j^*$ and go to Step 5. If $B(K) \ge B_l$, go to Step 4.

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

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

Step 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 B-infeasible policy is found, the set of switching points that can be increased or decreased at subsequent steps is reduced 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 presented in our previous work (Terekhov and Beck 2008) to solve the subproblem. 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 subproblem and master solutions are optimal. Otherwise, the CP method based on the *PSumsSubproblem* model and shaving is applied.

4.3.3. The *PSumsSubproblem* Model.

The *PSumsSubproblem* model has, as its decision variables, the set of k_i, $i = 0, 1, \dots, x+f$, each with an initial domain of $[0 \dots S]$. Additionally, it has two types of probability variables: $PSums(k_i)$ for $i = 0, \dots, x+f-1$, and $P(j)$ for $j = k_0, k_1, k_2, \dots, k_{x+f}$, 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 that ensures that the probability values satisfy the steady-state conditions of the frontroom 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 in the *PSums* model proposed by Terekhov and Beck (2008) for problem P_1.

$PSums(k_i)$ represents the probability of there being between k_i and $k_{i+1}-1$ customers in the front room and is defined in Equation (13). Equation (14) 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 $\beta Sum(k_i)$, $i = 1, \dots, x+f$, is defined in Equation (15).

$$PSums(k_i) = \begin{cases} P(k_i)\dfrac{1-[\lambda/((i+1)\mu)]^{k_{i+1}-k_i}}{1-\lambda/((i+1)\mu)} \\[1ex] \quad \text{if } \dfrac{\lambda}{((i+1)\mu)} \neq 1, \\[1ex] P(k_i)(k_{i+1}-k_i) \quad \text{otherwise,} \end{cases} \tag{13}$$

$$P(k_{i+1}) = \left[\frac{\lambda}{(i+1)\mu}\right]^{k_{i+1}-k_i} P(k_i), \tag{14}$$

$$\beta Sum(k_i) = \sum_{j=k_{i-1}+1}^{k_i} \beta_j = \begin{cases} X_i\left(\dfrac{\lambda}{\mu}\right)^{k_{i-1}-k_0+1}\left(\dfrac{1}{i}\right)\left[\dfrac{1-(\lambda/(i\mu))^{k_i-k_{i-1}}}{1-(\lambda/(i\mu))}\right] \\[1ex] \quad \text{if } \dfrac{\lambda}{i\mu} \neq 1, \\[1ex] X_i\left(\dfrac{\lambda}{\mu}\right)^{k_{i-1}-k_0+1}\left(\dfrac{l}{i}\right)(k_i-k_{i-1}) \\[1ex] \quad \text{otherwise.} \end{cases} \tag{15}$$

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

$$F_{\text{cross}} = \sum_{i=1}^{x} i[PSums(k_{i+f-1}) - P(k_{i+f-1}) + P(k_{i+f})]. \tag{16}$$

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

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

$$L = \sum_{i=0}^{x+f-1} L(k_i) + k_{x+f}P(k_{x+f}), \tag{17}$$

where

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

Expected customer waiting time is defined as in Equation (12).

The equations defining $PSums(k_i)$, $P(k_i)$, L, B, and W_q, together with $W_q \le W_u$ and $B \ge B_l$, are the major constraints of the CP model of the subproblem. Additionally, the set of constraints necessary for the definition of a policy (presented in §4.3.1) is included in this model.

4.3.4. Shaving. 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 (Demassey et al. 2005, van Dongen 2006). We use two shaving procedures: $B_lShaving$, which makes inferences based on the feasibility of policies with respect to the B_l constraint, and $W_uShaving$, which makes inferences based on feasibility with respect to the W_u constraint.

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 $W_q \le W_u$ is temporarily removed. At each step of the $B_lShaving$ procedure, $k_i = \min(k_i)$ or $k_i = \max(k_i)$ is temporarily added to the model for $i \in \{0, \ldots, x+f-1\}$. If $k_i = \min(k_i)$ is added, then all other switching points are assigned the maximum possible values subject to the condition that $k_n < k_{n+1}$, $\forall\, n \in \{0, \ldots, x+f-1\}$. 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 $\min(k_i)+1 \le \max(k_i)$, 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 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 satisfying the $W_q \le W_u$ constraint, and the subproblem 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 subproblem 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 $k_n < k_{n+1}$, $\forall\, n \in \{0, \ldots, x+f-1\}$. If the resulting policy is B-feasible but W_q-infeasible, the constraint $k_i < \max(k_i)$ can be permanently added, assuming $\max(k_i) - 1 \ge \min(k_i)$. Because 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 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 subproblem 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 subproblem solutions are optimal. The complete $B_lShaving$ algorithm is presented in Figure 1.

```
Algorithm 1: $B_lShaving$
Input: $S, \mu, \lambda, B_l, W_u$ (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
   subproblem 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 ($W_q < W_u$)
          return $policy$, $W_q$; stop, subproblem solution found
        if ($\max(Domain(k_i)) - 1 \ge \min(Domain(k_i))$)
          add($k_i < \max(Domain(k_i))$)
          $successfulShave = true$
          $changed = true$
        else
          stop, no policy with a better $W_q$ value exists;
            subproblem 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 ($W_q < W_u$)
          return $policy$, $W_q$; stop, subproblem solution found
      else
        if ($\min(Domain(k_i)) + 1 \le \max(Domain(k_i))$)
          add($k_i > \min(Domain(k_i))$)
          $successfulShave = true$
          $changed = true$
        else
          stop, no policy with a better $W_q$ value exists;
            subproblem is infeasible
      remove($k_i = \min(Domain(k_i))$)
```

Figure 1 $B_lShaving$ **Algorithm**

The $W_uShaving$ procedure makes inferences based strictly on the constraint $W_q \le W_u$. The constraint

```
Algorithm 2: $W_u Shaving$
Input: $S$, $\mu$, $\lambda$, $B_l$, $W_u$ (instance parameters); $x$, $f$, $b$ (current
  master problem solution)
Output: (possibly) modified domains of the variables $k_i$,
  or proof that subproblem 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;
            subproblem is infeasible
      remove($k_i = \max(Domain(k_i))$)
```

Figure 2 W_u*Shaving* **Algorithm**

$B \geq B_l$ 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. Because all switching 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 subproblem is proven. The $W_u Shaving$ algorithm is stated in Figure 2.

To solve the subproblem, 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.

4.4. Summary

Recall that we take a logic-based Benders' decomposition approach to our problem of finding the optimal staffing configuration. The master problem is presented as Equation (5). 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 subproblem, where the aim is to find a policy that simultaneously satisfies 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 $(x > x' \vee f > f' \vee b > b')$ is added to the master problem, and it is re-solved. The alternation between the master problem and the subproblem continues until a policy is found in the subproblem, which is both B-feasible and W_q-feasible, implying that the current master problem solution is optimal.

The subproblem 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 that satisfies the back room constraint, two cases may occur. First, the policy may also be W_q-feasible, in which case the current master solution is proven to be optimal. Second, the policy may not be able to satisfy the W_q constraint. In this case, there is no guarantee that the subproblem is infeasible, and we therefore use the *PSumsSubproblem* model with shaving and search. Shaving is composed of two procedures, $B_l Shaving$ and $W_u Shaving$, which are run iteratively until either feasibility 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 look for a feasible solution or show that none exists.

5. Experimental Results and Analysis I

Because 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. (Note: Numerical values in some of the results are slightly different from the ones presented in the previous work by Terekhov et al. 2007 because of some minor errors discovered after the publication of that paper. The main conclusions and analysis of the previous work by Terekhov et al. 2007 remain valid, however.) 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. It should be noted that the results presented below are 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 ILOG Solver precision to 0.000001 for all of our experiments.

5.1. Problem Size

Recall that S is the 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

Table 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

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
No. of iterations	4.57	7.77	16.07	6.13	8.00	7.23	8.23	34.73	27.60	23.83
Total no. 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

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 Terekhov and Beck (2008), for which they were generated in such a way as to ensure nontrivial 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. Because P_1 is similar to our subproblem, 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 1, the mean CPU time, the mean number of times the subproblem is solved (number of iterations), and the mean total number of workers in the optimal solution over 30 instances for each value of S are presented. 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} frontroom workers and B_{total} backroom workers (this solution can be obtained for all problem instances 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 subproblem. 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 subproblems (the maximum CPU time required to solve a subproblem is approximately 82.84 seconds). Moreover, the mean total number of workers for $S = 80$ is greater than for the other values of S because of the presence of several instances in this set that have tight constraints on the expected customer waiting time and the expected number of workers in the back room.

Although higher values of S lead to larger domain sizes for the k_is, and thus may result in higher run times, Table 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 $S = 30$ is higher than those at $S = 40$, 50, 60, and 70, and the mean CPU times at $S = 90$ and 100 are substantially lower than that at $S = 80$.

5.2. 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 one 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 1 also occurs under these five cost combinations, indicating that problem difficulty is a function of more than just the cost assumptions.

5.3. Other Parameters

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. (2005). Ten different cost combinations that satisfy our cost assumptions are used, giving us 540 instances. The cost combinations (c_x, c_f, c_b) used 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.

Table 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 $\lambda W_u/B_l$

	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
No. of iterations	1.00	3.20	11.30	1.00	5.70	22.58	3.20	14.10	36.50
Total no. 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

Although the values of the parameters vary, these instances can be grouped into nine types according to the value of $\lambda W_u/B_l$, which is an indication of how difficult it is 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 nine 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 2, it can be seen that as 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 $\lambda W_u/B_l$ 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.

5.4. Cost of the Optimal Solution

Finding the optimal mix of specialized and cross-trained workers is worthwhile only if there is a difference between the cost of the resulting solution and the costs of the two "naive" solutions, the specialized-only solution and the cross-trained-only solution. We investigate this question by comparing the staffing costs obtained using these three methods for the same set of 540 problem instances.

The specialized-only solution $f = F_{\text{total}}$, $b = B_{\text{total}}$, $x = 0$ can be easily determined using the equations of §4.1. The cross-trained-only solution $f = 0$, $b = 0$, $x = X_{\text{total}}$, where X_{total} is the smallest number of cross-trained workers required in the facility to satisfy both constraints, is a solution to the Berman et al. (2005) problem P_2 (described in §2). To obtain the value of X_{total}, 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 (5)) and ran it with costs $c_x = 1$, $c_f = 100$, $c_b = 100$. However, our method was unable to solve 19 of the 300 instances discussed in §5.1 within 600 seconds. In these cases, we assume that the best cross-trained-only solution consists of $F_{\text{total}} + B_{\text{total}}$ cross-trained workers, and our results are slightly inaccurate.

Tables 1 and 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. To determine these values, 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 nonzero difference between the costs of the optimal solution and the two "naive" solutions, indicating that our approach may be valuable in practical applications.

6. Experimental Results and Analysis II

The subproblem of our logic-based Benders' decomposition method, described in §4.3, 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 that are modifications of our original method:

- *heuristicOnly*, in which the solution to the subproblem can be found only by the heuristic (shaving and search are removed);
- *noShaving*, in which the subproblem 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 subproblem (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 subproblem solution, the subproblem is truly infeasible. In particular, the heuristic can guarantee infeasibility of the subproblem only if the policy $\widehat{\widehat{K}}$ violates the backroom constraint or the policy $\widehat{K}$ violates the frontroom 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 $\widehat{K}$ is infeasible with respect to the frontroom constraint, or $\widehat{\widehat{K}}$ is infeasible for the backroom 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 subproblem 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 $\widehat{K}$ is feasible for the frontroom constraint and $\widehat{\widehat{K}}$ is feasible for the backroom constraint, but the heuristic is unable to find a feasible policy, search is run to prove the feasibility or infeasibility of the subproblem. However, if this method takes longer than the allowed run-time limit, we assume that the solution it returns is the specialized-only solution (because 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 §4.3) on the set of 540 instances from §5.3.

6.1. Solution Quality

Experiments with our set of 540 instances indicate that the *heuristicOnly* method finds an optimal-cost 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, finds the optimal-cost staff mix in all 540 instances.

In Table 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, Table 3 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 subproblem, 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), 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.

Table 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 $\lambda W_u/B_l$

	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	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 Mean Run Times for ***heuristicOnly***, ***noSearch***, ***noShaving***, and Our Original Method for Each Value of $\lambda W_u/B_l$

	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	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

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.

6.2. Mean Run Time

Table 4 presents the mean run times for the *heuristicOnly*, the *noSearch*, and the *noShaving* methods, and our original method. The only method of these four that is unable to solve all 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 underestimates of the true means.

We can first observe that the mean run times of our original method most closely resemble those of the *noSearch* method. Second, 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 $\lambda W_u/B_l$. This is not surprising, since the original heuristic of Berman et al. (2005) has been shown by Terekhov and Beck (2008) to be extremely fast for problem P_1.

Third, the longest run times result from the *noShaving* method, since search has to be used in all cases for which the heuristic is unable to guarantee infeasibility and also unable to find a feasible solution. In previous work (Terekhov and Beck 2008), 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 subproblem 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 subproblem 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 $S = 80$ in experiments of §§5.1 and 5.2) 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.

6.3. Mean Number of Iterations

In Table 5, we see that the mean number of iterations for all but one value of $\lambda W_u/B_l$ 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 $\lambda W_u/B_l = 21.87$ is because, as shown in §6.1, 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 instances where shaving is not able to prove feasibility of the subproblem given the optimal master solution, it is (incorrectly) assumed that the subproblem 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 subproblem in these cases, resulting in a smaller number of iterations.

Table 5 **Mean Number of Iterations for *heuristicOnly*, *noSearch*, *noShaving*, and Our Original Method for Each Value of $\lambda W_u/B_l$**

	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

We should also note that the *noShaving* method has a smaller or equivalent number of iterations compared with the other methods. This is because using search to solve each subproblem 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 of the four methods for all values of λ/μ. This is because the heuristic misses some feasible solutions, and as a result, the method has to consider more master solutions, increasing the total number of iterations.

6.4. Summary

Overall, these results indicate, first, 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 subproblem 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.

Second, we see that the heuristic is a helpful component of our method, because, in most instances, it allows us to quickly guarantee the feasibility of the subproblem 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, because the *noSearch* method is able to guarantee optimality in most cases (see Table 3), search is seldom used. When it is needed for guaranteeing completeness, it usually results in the instance not being solved within 600 seconds.

7. Further Work

In this paper, the operational question of when worker switching should occur was addressed by determining an a priori policy that ensures satisfactory steady-state performance of the facility. However, switching of workers may be viewed as an online decision. This observation raises the question of whether the problem could be solved using, for example, online stochastic combinatorial optimization (OSCO) (Van Hentenryck and Bent 2006). Investigation of the applicability of both the modelling and computational approaches from OSCO to the problem of worker switching is one possible direction for further work.

Future work may also focus on further integration of queueing theory and constraint programming with the goal of solving problems that have both a complex combinatorial structure for which CP is known to work well, and a stochastic nature that can be effectively modelled using queueing theory.

8. Conclusions

In this paper, a logic-based 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. 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 subproblem 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 subproblem quickly.

Because the problem discussed is a queueing design and control problem, this paper demonstrates that constraint programming may be a useful approach for solving such problems. Thus, this work provides one of the first links between queueing theory and CP.

Acknowledgments

This work has received support from the Science Foundation Ireland under Grant 00/PI.1/C075, the Natural Sciences and Engineering Research Council of Canada, Canada Foundation for Innovation, and ILOG, SA. A preliminary version of parts of this work has been previously published (Terekhov et al. 2007).

References

Baptiste, P., P. Laborie, C. Le Pape, W. Nuijten. 2006. Constraint-based scheduling and planning. F. Rossi, P. van Beek, T. Walsh, eds. *Handbook of Constraint Programming*, Chapter 22. Elsevier, Amsterdam, 761–799.

Batta, R., O. Berman, Q. Wang. 2007. Balancing staffing and switching costs in a service center with flexible servers. *Eur. J. Oper. Res.* **177**(2) 924–938.

Benders, J. F. 1962. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* **4**(1) 238–252.

Berman, O., R. C. Larson. 2004. A queueing control model for retail services having backroom operations and cross-trained workers. *Comput. Oper. Res.* **31**(2) 201–222.

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

Berman, O., J. Wang, K. P. Sapna. 2005. Optimal management of cross-trained workers in services with negligible switching costs. *Eur. J. Oper. Res.* **167**(2) 349–369.

Brown, K. N., I. Miguel. 2006. Uncertainty and change. F. Rossi, P. van Beek, T. Walsh, eds., *Handbook of Constraint Programming*, Chapter 21. Elsevier, Amsterdam, 731–760.

Brusco, M. J. 2008. An exact algorithm for a workforce allocation problem with application to an analysis of cross-training policies. *IIE Trans.* **40**(5) 495–508.

Cezik, M. T., P. L'Ecuyer. 2008. Staffing multiskill call centers via linear programming and simulation. *Management Sci.* **54**(2) 310–323.

Chevalier, P., N. Tabordon. 2003. Overflow analysis and cross-trained servers. *Internat. J. Prod. Econom.* **85**(1) 47–60.

Demassey, S., C. Artigues, P. Michelon. 2005. Constraint-propagation-based cutting planes: An application to the resource-constrained project scheduling problem. *INFORMS J. Comput.* **17**(1) 52–65.

Fox, M. S. 1983. Constraint-directed search: A case study of job-shop scheduling. Ph.D. thesis, CMU-RI-TR-85-7, Intelligent Systems Laboratory, The Robotics Institute, Carnegie Mellon University, Pittsburgh.

Gross, D., C. Harris. 1998. *Fundamentals of Queueing Theory*. John Wiley & Sons, New York.

Hooker, J. N., G. Ottosson. 2003. Logic-based Benders' decomposition. *Math. Programming* **96**(1) 33–60.

Tadj, L., G. Choudhury. 2005. Optimal design and control of queues. *TOP* **13**(2) 359–412.

Tarim, S. A., I. Miguel. 2005. A hybrid Benders' decomposition method for solving stochastic constraint programs with linear recourse. B. Hnich, M. Carlsson, F. Fages, F. Rossi, eds. *Recent Advances in Constraints*: *Joint ERCIM/CoLogNET Internat. Workshop on Constraint Solving and Constraint Logic Programming*. *Lecture Notes in Artificial Intelligence*, Vol. 3978. Springer, Heidelberg, Germany, 133–148.

Terekhov, D., J. C. Beck. 2008. A constraint programming approach for solving a queueing control problem. *J. Artificial Intelligence Res.* **32** 123–167.

Terekhov, D., J. C. Beck. 2009. An extended queueing control model for facilities with front room and back room operations and mixed-skilled workers. *Eur. J. Oper. Res.* **198**(1) 223–231.

Terekhov, D., J. C. Beck, K. N. Brown. 2007. Solving a stochastic queueing design and control problem with constraint programming. *Proc. 22nd Conf. Artificial Intelligence* (*AAAI'07*), *Vancouver*, Association for the Advancement of Artificial Intelligence, Menlo Park, CA, 261–266.

van Dongen, M. R. C. 2006. Beyond singleton arc consistency. *Proc. 17th Eur. Conf. Artificial Intelligence* (*ECAI'06*), *Riva del Garda*, IOS Press, Amsterdam, 163–167.

Van Hentenryck, P., R. Bent. 2006. *Online Stochastic Combinatorial Optimization*. MIT Press, Cambridge, MA.