Les Cahiers du GERAD ISSN: 0711–2440

Staffing Multiskill Call Centers via Linear Programming and Simulation

M.T. Cezik
P. L'Ecuyer

G–2004–97

December 2004

Staffing Multiskill Call Centers via Linear Programming and Simulation

Mehmet Tolga Cezik, Pierre L'Ecuyer

GERAD and Département d'Informatique et de Recherche Opérationnelle
Université de Montréal
C.P. 6128, Succ. Centre-ville
Montréal (Québec) Canada H3C 3J7

December 2004

Les Cahiers du GERAD

G–2004–97

Abstract

We study an iterative cutting-plane algorithm on an integer program, for minimizing the staffing costs of a multiskill call center subject to service-level requirements which are estimated by simulation. We solve a sample average version of the problem, where the service-levels are expressed as functions of the staffing for a fixed sequence of random numbers driving the simulation. An optimal solution of this sample problem is also an optimal solution to the original problem when the sample size is large enough. Several difficulties are encountered when solving the sample problem, especially for large problem instances, and we propose practical heuristics to deal with these difficulties. We report numerical experiments with examples of different sizes. The largest example corresponds to a real-life call center with 65 types of calls and 89 types of agents (skill groups), and we find a good solution it in about half an hour.

Résumé

Nous développons un algorithme itératif basé sur des plans de coupe appliqué à un programme d'optimisation en nombres entiers avec contraintes non-linéaires, dont le but est d'optimiser l'allocation des agents de différents types dans un centre d'appels "multiskill", sous des contraintes sur le niveau de service. Nous résolvons une version empirique ("moyenne échantillonnage") de ce problème où les niveaux de service sont exprimés comme des fonctions du vecteur d'allocation des agents pour une réalisation aléatoire fixée (i.e., une suite fixée de valeurs aléatoires uniformes). Ces fonctions sont calculées par simulation en utilisant des valeurs aléatoires communes. Une solution optimale de la version empirique du problème est aussi une solution optimale du problème original lorsque la taille échantillonnale est assez grande. La résolution du problème empirique devient très difficile lorsque ce dernier est de grande taille. Nous proposons des heuristiques pratiques pour contourner les difficultés et obtenir de bonnes solutions en temps raisonnable. Nous donnons des résultats d'expériences numériques pour des exemples de différentes tailles. L'un de ces exemples correspond à un centre d'appels réaliste recevant 65 types d'appels et ayant 89 types d'agents. Nous trouvons une bonne solution en une demi-heure environ.

Acknowledgments: This research was supported by grants number OGP-0110050 and CRDPJ-251320 from NSERC-Canada, a grant from Bell Canada (via the "Laboratoires Universitaires Bell"), grant number 02ER3218 from FQRNT-Québec, and a Canada Research Chair to the second author. We thank Éric Buist, who implemented the call center simulation package, Wyean Chan, who helped with the programming, and Athanassios Avramidis, who gave several insightful comments.

1 Introduction

We consider a telephone *call center*, or a more general *contact center*, where different types of calls arrive at random and different groups of agents answer these calls. Each type of call requires a specific *skill* and each agent group (also called *skill group*) has a given subset of these skills, so all agents in that group can handle the corresponding call types and only those. There may be preferences among these agent groups for a given type of call, either because some groups are better than others at handling it or because we prefer to save certain agent groups for other call types.

The calls arrive according to arbitrary stochastic processes that could be nonstationary, and perhaps doubly stochastic (see, e.g., Avramidis et al., 2004). An arriving call can be served immediately if an agent with the appropriate skill is available, or may have to wait in a queue. An *abandonment* occurs whenever the waiting time of a call exceeds its (random) *patience time*. That call is then lost. *Skill-based routing* (SBR) strategies are used to determine which agent group handles each individual call.

The goal is to minimize the operating cost of the center under a set of constraints on the quality of service (QoS). The decisions to be made are how many agents of each skill group to have in the center as a function of time. In a *staffing* problem, the day is divided into periods (e.g., 30 minutes or one hour each) and one simply decides the number of agents of each group for each period. In a *scheduling* problem, a set of admissible work schedules is first specified, and the decision variables are the number of agents of each skill group in each work schedule. This determines the staffing indirectly, while making sure that it corresponds to a feasible set of work schedules. A yet more restrictive version of the problem is when there is a fixed set of available agents to be scheduled for the day or the week, where each agent has a specific set of skills. Then we have a *scheduling and rostering* problem.

In this paper, the objective function is the sum of costs of all agents, where the *cost* of an agent depends on its set of skills. This cost is typically modelled as a constant plus an increment of between five to twenty percent for each additional skill. The QoS *constraints* are that the fraction of calls answered within a certain time limit, in the long run, exceeds a given threshold. This fraction is called the *service-level*. In general we may also have an upper bound on the abandonment ratio (the fraction of calls that are lost), or on other types of performance measures. Such constraints can be imposed per call type, per period, and globally, with different thresholds. In this paper, we consider only service-level constraints.

If each agent group has a single skill, we have a system of separate parallel queues, one queue per call type. If there is a single agent group with all skills, we have a single queue and (other things being equal) smaller waiting times than with separate queues. However, agents with more skills are more costly, so there is a compromise to be made in choosing the skill groups. Wallace and Whitt (2004) have shown empirically, in a certain setting and for properly balanced systems with good routing strategies, that one or two skills per agent often gives a performance (in terms of QoS) that is almost as good as for a system where all agents have all skills.

In a more general formulation, we would also want to optimize the routing of the calls. In *static routing*, each call type may have an ordered list of agent groups to try. If all are busy, the call joins a queue. There could be one queue per agent group, or one queue per call type (this is usually better, because it allows more resource sharing), or one single queue for several call types, or a mixture of these. Priorities and complex routing rules can thus be implemented. For example, each agent may have an ordered list of queues to dig from when it becomes available, and the order is not necessarily the same for all agents in a group. The priorities may change based on thresholds on queue sizes or waiting times, etc. In a general *dynamic routing* scheme, the decisions may depend on the entire state of the system, which may include the current time, the number of calls of each type in service and in the queues, the elapsed service time of the calls in service, etc. Optimal dynamic schemes are usually too complicated and difficult to implement, so in practice the routing strategies are selected from a prespecified class of simpler rules. In the optimization problems studied in this paper, they are assumed to be fixed and we optimize only the staffing or scheduling. This was motivated by a specific request from an industry who wanted precisely this kind of optimization tool.

Figure 1 gives a schematic view of arrival, routing, and service in a general multiskill call center with K call types and I agent groups. The λ_k's represent the arrival rates for the different call types, the servers S_i represent the skill groups, and the mean service times $1/\mu_{k,i}$ may depend jointly on the call type and skill group.

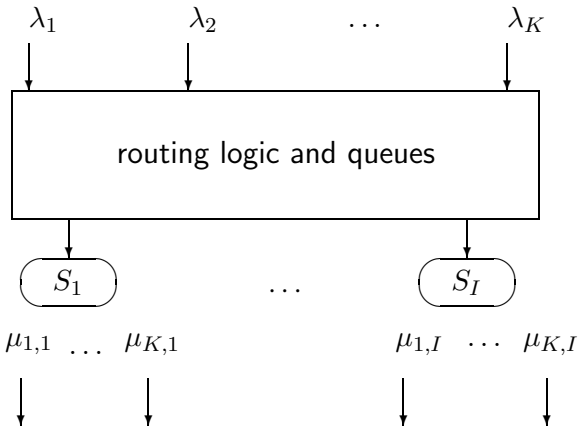

Figure 1: A General Multiskill Call Center

For general background on call center management, staffing and scheduling, and multi-skill centers with SBR, we refer the reader to Gans et al. (2003); Koole and Mandelbaum (2002); Wallace and Whitt (2004).

Atlason et al. (2004) have proposed a general methodology, based on the cutting plane method of Kelley Jr. (1960), to optimize the scheduling of agents in a single-call-type and single-skill call center, under service-level contraints. Their method combines simulation with integer programming and cut generation. The general idea is to optimize a relaxation

of a *sample average* version of the problem (which becomes a deterministic problem) by generating cuts from the violated service-level constraints and adding corresponding linear constraints until the optimal solution of the relaxed problem is feasible for the original problem. They solved an example with 5 periods and 6 different types of work schedules (i.e., 6 integer-valued decision variables).

Our aim in this paper is to extend their methodology to the multiskill setting, explore the difficulties encountered with larger problem instances, and develop (heuristic) methods to deal with these problems in a practical way. Some of the difficulties encountered are specific to the multiskill setting. With our improved methodology, we can solve much larger problem instances than with the original methodology of Atlason et al. (2004).

The remainder of the paper is organized as follows. In Section 2, we formulate the staffing and scheduling problems in a multiskill center. In Section 3, we first outline the solution methodology, based on simulation and cut generation. Then we explain several difficulties encountered in applying this methodology and how we get around them to solve realistic problems. In Section 4, we report on numerical experiments for staffing problems over a single period, in which we assume that the system is in steady-state. We solve problems of various sizes, from an artificial example with 5 call types and 12 skill groups to an example of a large real-life call center with 65 call types and 89 skill groups.

Staffing a call center in steady-state is not quite the same as the full scheduling problem (P1) that we formulate in the next section. However, solving large instances of it is a first step towards solving large instances of (P1). We have also solved scheduling problems with several periods, with more than 100 types of work schedules, but only two types of agents, in the context of a call center operating in blend mode (with inbound and outbout calls), with the methodology described here. The results will be reported elsewhere.

2 Problem Formulation

We now give a nonlinear integer programming formulation of a scheduling problem with service-level constraints, over a time interval divided into periods. There are K call types, I skill groups, P periods, and Q types of work schedules (which we also call *shifts*). For example, the scheduling could be for one day (or week) and the periods could be successive half hours of operation of the center over that day (or week). Each shift specifies the time when the agent starts working, when he/she finishes, and all the lunch and coffee breaks.

The *cost vector* is $\mathbf{c} = (c_{1,1}, \ldots, c_{1,Q}, \ldots, c_{I,1}, \ldots, c_{I,Q})^\mathsf{t}$, where $c_{i,q}$ is the cost of an agent of type i having shift q. The vector of *decision variables* is $\mathbf{x} = (x_{1,1}, \ldots, x_{1,Q}, \ldots, x_{I,1}, \ldots, x_{I,Q})^\mathsf{t}$, where is $x_{i,q}$ the number of agents of type i having shift q. We use the vector of *auxiliary variables* $\mathbf{y} = (y_{1,1}, \ldots, y_{1,P}, \ldots, y_{I,1}, \ldots, y_{I,P})^\mathsf{t}$ where $y_{i,p}$ is the number of agents of type i in period p. This vector $\mathbf{y}$ satisfies $\mathbf{y} = \mathbf{A}\mathbf{x}$ where $\mathbf{A}$ is a block diagonal matrix with I blocks $\tilde{\mathbf{A}}$, where the element (p, q) of $\tilde{\mathbf{A}}$ is 1 if shift q covers period p, and 0 otherwise.

The constraints are on the service-level, defined as the fraction of calls answered within a specified time limit. This time limit was taken as 20 seconds in our computational experiments. The *service-level* for call type k in period p is defined as

$$g_{k,p}(\mathbf{y}) = \frac{E[\text{num. of calls answered within } s_{k,p} \text{ seconds in period } p]}{E[\text{num. of calls in period } p]}$$

for some constant $s_{k,p}$. It is equal (with probability one) to the fraction of calls of type k answered within $s_{k,p}$ seconds over an infinite number of i.i.d. copies of period p. This fraction generally depends on the staffing over all periods, up to period p, because reducing the staffing over a given period can increase the queue lengths and thus reduce the service-level in the periods that follow, and may also increase the waiting time of calls answered in this given period but that arrived in earlier periods. This is why $g_{k,p}$ is written as a function of $\mathbf{y}$. In our model, the abandonments that occur before the time limit $s_{k,p}$ are not counted, whereas the calls that abandon after the time limit are counted in the total. The service-level constraints are of the form $g_{k,p}(\mathbf{y}) \geq l_{k,p}$ where each constant $l_{k,p}$ belongs to the interval $[0, 1)$.

The model also has constraints on the aggregate service-level over a given call type, a given period, and overall. For example, the aggregate service-level over call type k is the expected total number of calls of type k answered within some time limit s_k over the day (say), divided by the expected total number of calls of type k received over the day. We denote by $g_p(\mathbf{y})$, $g_k(\mathbf{y})$ and $g(\mathbf{y})$ the aggregated service-levels for period p, call type k, and overall, respectively. The corresponding time limits are s_p, s_k, and s, and the corresponding minimal service-levels are l_p, l_k and l.

These functions $g_\bullet$ (the ratios of expectations) are generally unknown and much too complicated to be evaluated exactly. They can be either *approximated* via simplified queueing models (e.g., as in Ingolfsson and Cabral, 2002) or *estimated* by simulation. In this paper, we concentrate on the second alternative.

With this notation, we now define the *scheduling problem* as

$$\begin{aligned}
\min \quad & \mathbf{c}^{\mathrm{t}}\mathbf{x} = \textstyle\sum_{i=1}^{I} \sum_{q=1}^{Q} c_{i,q} x_{i,q} \\
\text{subject to} \quad & \mathbf{A}\mathbf{x} = \mathbf{y}, \\
& g_{k,p}(\mathbf{y}) \geq l_{k,p} \qquad \text{for all } k, p, \\
& g_p(\mathbf{y}) \geq l_p \qquad \text{for all } p, \\
& g_k(\mathbf{y}) \geq l_k \qquad \text{for all } k, \\
& g(\mathbf{y}) \geq l, \\
& \mathbf{x} \geq 0, \text{ and integer.}
\end{aligned} \tag{P1}$$

The *staffing Problem* is a *relaxation* of the scheduling problem where we forget about the admissibility of schedules and just assume that any staffing is admissible. The cost

vector in this setting is $\mathbf{c} = (c_{1,1}, \ldots, c_{1,P}, \ldots, c_{I,1}, \ldots, c_{I,P})^{\mathsf{t}}$ where $c_{i,p}$ is the cost of an agent of group i in period p. We have:

$$\begin{array}{rl} \min & \mathbf{c}^{\mathsf{t}}\mathbf{y} = \sum_{i=1}^{I} \sum_{p=1}^{P} c_{i,p} y_{i,p} \\ \text{subject to} & g_{k,p}(\mathbf{y}) \geq l_{k,p} \qquad \text{for all } k, p, \\ & g_p(\mathbf{y}) \geq l_p \qquad \text{for all } p, \\ & g_k(\mathbf{y}) \geq l_k \qquad \text{for all } k, \\ & g(\mathbf{y}) \geq l, \\ & \mathbf{y} \geq 0, \text{ and integer.} \end{array} \tag{P2}$$

In the special case where we consider *one period at a time*, we have $\mathbf{c} = (c_1, \ldots, c_I)^{\mathsf{t}}$ where c_i is the cost of an agent of type i and $\mathbf{y} = (y_1, \ldots, y_I)^{\mathsf{t}}$ where y_i is the number of agents of type i. In this context, we often assume that the system is in *steady-state* over the given period (but we may also assume arbitrary initial conditions). The optimization problem then simplifies to:

$$\begin{array}{rl} \min & \mathbf{c}^{\mathsf{t}}\mathbf{y} = \sum_{k=1}^{K} c_k y_k \\ \text{subject to} & g_k(\mathbf{y}) \geq l_k \qquad \text{for all } k, \\ & g(\mathbf{y}) \geq l, \\ & \mathbf{y} \geq 0, \text{ and integer.} \end{array} \tag{P3}$$

Our numerical illustrations in Section 4 are with problem (P3) for a steady-state system. Solving this problem in itself can bring useful insight into the effect of skill set selection and routing strategies on the service-level.

It is frequent practice in call centers to solve the single-period staffing problem independently for each period, under a steady-state assumption (first step), and then find an admissible schedule that covers the required staffing at minimal cost (second step). This two-step approach is definitely suboptimal and the suboptimality gap can be significant (e.g., Jennings et al., 1996) but it is nevertheless often used because the original scheduling problem (P1) is deemed too hard to solve. If the staffing vector $\mathbf{y}^*$ represents the optimal solution obtained in the first step, then the problem to solve in the second step is

$$\begin{array}{rl} \min & \mathbf{c}^{\mathsf{t}}\mathbf{x} = \sum_{i=1}^{I} \sum_{q=1}^{Q} c_{i,q} x_{i,q} \\ \text{subject to} & \mathbf{A}\mathbf{x} \geq \mathbf{y}^*, \\ & \mathbf{x} \geq 0, \text{ and integer.} \end{array} \tag{P4}$$

To solve any of these problems, we need to approximate or estimate the functions $g_\bullet$. For this, we tried variations of a queueing approximation proposed by Koole and Talim (2000) and Koole et al. (2003), where the model is approximated by a loss system (with several simplifications) and a heuristic is used to infer the service-levels in the original model from the loss ratios in the loss model. We compared the results with those of a detailed simulation model and found significant differences in the service-levels (more than 100% in some examples). In the remainder of this paper, we use only simulation to estimate the service-level.

3 General Methodology

3.1 Optimization of a Sample Problem

The service-levels involved in the constraints are estimated by simulation. Suppose we simulate the center n times, independently, over its P periods of operation. Let ω represent the source of *randomness*, i.e., the sequence of all independent $U(0,1)$ random variates that drive the successive simulation runs (regardless of their number). We may assume that ω is divided into long disjoint subsequences so that the ith simulation run always starts using random numbers at the beginning of the ith subsequence regardless of the model parameters and staffing vector. This is implemented by using random number packages that provide multiple streams and substreams (L'Ecuyer et al., 2002; L'Ecuyer, 2004).

The *empirical service-level* over n simulation runs, defined as the observed number of calls answered within the time limit divided by the total number of calls, is a function of the staffing level $\mathbf{y}$ and of ω. We denote it by $G_{n,k,p}(\mathbf{y},\omega)$ for call type k in period p; $G_{n,p}(\mathbf{y},\omega)$ aggregated over period p; $G_{n,k}(\mathbf{y},\omega)$ aggregated for call type k; and $G_n(\mathbf{y},\omega)$ aggregated overall. For a *fixed* ω, these are all deterministic functions of $\mathbf{y}$. To compute these functions at different values of $\mathbf{y}$, we simply use simulation with *common random numbers*, i.e., make sure that the same random numbers are used at the same place for all values of $\mathbf{y}$ (Law and Kelton, 2000).

If we replace the functions $g_\bullet$ in the optimization problems by their sample counterparts $G_\bullet$, we obtain the following *empirical scheduling problem*, which is a *sample average* version of the original problem:

$$
\begin{array}{rl}
\min & \mathbf{c}^{\mathrm{t}}\mathbf{x} = \sum_{i=1}^{I}\sum_{q=1}^{Q} c_{i,q}x_{i,q} \\
\text{subject to} & \mathbf{A}\mathbf{x} = \mathbf{y}, \\
& G_{n,k,p}(\mathbf{y}) \geq l_{k,p} \qquad \text{for all } k,p, \\
& G_{n,p}(\mathbf{y}) \geq l_p \qquad \text{for all } p, \\
& G_{n,k}(\mathbf{y}) \geq l_k \qquad \text{for all } k, \\
& G_n(\mathbf{y}) \geq l, \\
& \mathbf{x} \geq 0, \text{ and integer.}
\end{array}
\tag{P5}
$$

A similar formulation can be given for the other problems, replacing g by G everywhere. The single-period staffing problem becomes:

$$\begin{array}{rl} \min & \mathbf{c}^{\mathrm{t}}\mathbf{y} = \sum_{k=1}^{K} c_k y_k \\ \text{subject to} & G_{n,k}(\mathbf{y}) \geq l_k \qquad \text{for all } k, \\ & G_n(\mathbf{y}) \geq l, \\ & \mathbf{y} \geq 0, \text{ and integer.} \end{array} \tag{P6}$$

We insist on the fact that when ω is fixed, these sample problems become purely deterministic. These are the problems we solve, instead of the original problems (P1) to (P3).

We can study the convergence of the optimal solution of the sample problem to that of the original problem as $n \to \infty$, under the assumption that ω is really an infinite sequence of i.i.d. random variables (as opposed to pseudorandom numbers). This was done by Atlason et al. (2004) for their model, and earlier, e.g., by Vogel (1994) in a more general setting. Our argumentation is similar but simplified.

In our context, it is reasonable to assume that the set $\mathcal{Y}$ of potential solutions $\mathbf{y}$ is *finite* (e.g., it suffices to impose an upper bound on the total number of agents in the center). Let $\mathcal{Y}^*$ be the set of optimal solutions of the *exact* problem. Another reasonable assumption is that no service-level constraint is satisfied *exactly* in the original problem for these solutions $\mathbf{y} \in \mathcal{Y}^*$. Let $\mathcal{Y}_n^*$ be the set of optimal solutions of the *sample* problem based on n simulation runs. Each random variable $G_{n,k,p}(\mathbf{y}, \omega)$ estimates the ratio of expectations $g_{k,p}(\mathbf{y})$ by the ratio of two averages of n i.i.d. random variables having the correct (positive) expectations and finite variance. Therefore $G_{n,k,p}(\mathbf{y}, \omega)$ converges to $g_{k,p}(\mathbf{y})$ with probability one when $n \to \infty$. This means that for every $\epsilon > 0$, there is a random variable $N_0 < \infty$ such that $|G_{n,k,p}(\mathbf{y}, \omega) - g_{k,p}(\mathbf{y})| < \epsilon$ for all $n \geq N_0$. Let δ be the smallest absolute difference between an exact service-level $g_\bullet$ and its corresponding threshold $l_\bullet$, among all solutions $\mathbf{y} \in \mathcal{Y}$. In view of the above assumptions, since $\mathcal{Y}$ is finite, there is a random N_* such that $|G_{n,k,p}(\mathbf{y}, \omega) - g_{k,p}(\mathbf{y})| < \delta$ for all $n \geq N_*$, all $\mathbf{y} \in \mathcal{Y}$, all k, all p, and all aggregate service-levels as well. Then, for all $n \geq N_*$, the sample problem has exactly the same set of feasible solutions (and therefore the same optimal solutions) as the exact problem. We have just proved the following:

Proposition 1 *With probability 1, there is an integer $N_* < \infty$ such that for all $n \geq N_*$, $\mathcal{Y}_n^* = \mathcal{Y}^*$.*

The next proposition gives a convergence result in the sense of large deviation theory, under the assumption that the service-level estimators satisfy a standard large deviation principle (see, e.g., Ellis, 1985; Shwartz and Weiss, 1995). This assumption holds under fairly general conditions that should practically always hold in our setting. It typically holds with $\kappa = \kappa_0 \epsilon^2$ for every ϵ small enough, for some constant $\kappa_0 > 0$.

Assumption 1 *For every $\epsilon > 0$, there are positive integers n_0 and κ such that for all $n \geq n_0$ and $\mathbf{y} \in \mathcal{Y}$,*

$$P\left(|G_{n,k,p}(\mathbf{y},\omega) - g_{k,p}(\mathbf{y})| > \epsilon\right) \leq e^{-n\kappa}$$

for all k, p, and for the aggregate service-levels as well.

Proposition 2 *Under Assumption 1, there is a positive constant α and an integer $n_* < \infty$ such that for all $n \geq n^*$,*

$$P[\mathcal{Y}_n^* = \mathcal{Y}^*] \geq 1 - \alpha e^{-n\kappa}.$$

Proof. Take $\epsilon = \delta$ in the assumption. If the absolute error $|G_{n,k,p}(\mathbf{y},\omega) - g_{k,p}(\mathbf{y})|$ is less than δ simultaneously for all constraints and all $\mathbf{y} \in \mathcal{Y}$, then the sample problem has exactly the same feasible set as the exact problem and therefore $\mathcal{Y}_n^* = \mathcal{Y}^*$. But the probability that this does not happen is bounded by the sum over all constraints and all values of $\mathbf{y}$ of the probabilities that it does not happen for this particular constraint and value of $\mathbf{y}$. Each such probability is bounded by $e^{-n\kappa}$ from the assumption and there are a finite number of them, so their sum must be bounded by $\alpha e^{-n\kappa}$ for some constant α. □

The proof of the proposition also implies that the sample and exact problems have exactly the same set of feasible solutions with a probability that converges to 1 exponentially fast when $n \to \infty$. It reuses an argument form the proof of Proposition 2 of Yakowitz et al. (2000).

For solving the staffing problem (P3) in the *steady-state* case, we use the sample problem (P6) where the functions $G_{n,k}$ and G_n are estimated by simulating the model for n hours of operation. These functions converge to g_k and g and obey a central-limit theorem (pointwise) when $n \to \infty$ in the same way as for the multi-period finite-horizon model, if we assume that the system has enough ergodicity. Then, if we restrict the search to a finite subset of the space of stable solutions $\mathbf{y}$, Propositions 1 and 2 hold in this case as well.

3.2 Integer Linear Programming with Cut Generation

We solve the sample staffing and scheduling problems by linear programming with cut generation, in a manner similar to Atlason et al. (2004). The idea is to relax the nonlinear service-level constraints and add progressively linear constraints until the optimal solution satisfies all service-level constraints.

We give the following explanations on cut generation with the global service-level function g, to simplify the notation. The same reasoning applies as well to the service-level per call type and/or per period, and to their sample counterparts.

At any given step of the algorithm, let $\bar{\mathbf{y}}$ denote the current solution (optimal for the relaxed problem with the current set of constraints). If $\bar{\mathbf{y}}$ satisfies all service-level constraints, then it is an optimal feasible solution for our problem and we are done (assuming that we did not cut out the optimal solution(s) when adding linear constraints). Otherwise, take a violated constraint, say $g(\bar{\mathbf{y}}) < l$, and *suppose* that g is *concave in* $\mathbf{y}$. (A discrete function

g is called *concave* if the upper boundary of the convex hull of the points $(\mathbf{y}, g(\mathbf{y}))$ is a concave function.) Let $\bar{\mathbf{q}}$ be any *subgradient* of g at $\bar{\mathbf{y}}$, i.e., a vector that satisfies

$$g(\mathbf{y}) \leq g(\bar{\mathbf{y}}) + \bar{\mathbf{q}}^{\mathsf{t}}(\mathbf{y} - \bar{\mathbf{y}}) \tag{1}$$

for all $\mathbf{y}$. We want $g(\mathbf{y}) \geq l$, so we must have $l \leq g(\mathbf{y}) \leq g(\bar{\mathbf{y}}) + \bar{\mathbf{q}}^{\mathsf{t}}(\mathbf{y} - \bar{\mathbf{y}})$, i.e.,

$$\bar{\mathbf{q}}^{\mathsf{t}}\mathbf{y} \geq \bar{\mathbf{q}}^{\mathsf{t}}\bar{\mathbf{y}} + l - g(\bar{\mathbf{y}}), \tag{2}$$

where the term on the right is a constant that can be readily computed. Adding this linear *cut inequality* to the constraints removes $\bar{\mathbf{y}}$ from the current set of feasible solutions, without removing any solution that is feasible for the original problem if g is concave.

Unfortunately, g is not always concave in our setting. Typically, it is concave in the areas where $\mathbf{y}$ is large, but not where $\mathbf{y}$ is small (see the next section). But it is quite reasonable to assume at least that g is non-decreasing. In that case, a subgradient $\bar{\mathbf{q}}$ must necessarily satisfy $\bar{\mathbf{q}} \geq \mathbf{0}$, because taking $\mathbf{y} = \bar{\mathbf{y}} + \mathbf{e}_j$ in (1) gives $\bar{q}_j \geq g(\mathbf{y}) - g(\bar{\mathbf{y}}) \geq 0$, for each j, and the next proposition provides a *sufficient* condition ensuring that constraint (2) cannot remove feasible solutions of the original problem.

Proposition 3 *Suppose that g is nondecreasing everywhere in its domain and concave in the area (slice) Δ defined by the two linear inequalities*

$$\bar{\mathbf{q}}^{\mathsf{t}}\bar{\mathbf{y}} \geq \bar{\mathbf{q}}^{\mathsf{t}}\mathbf{y} \geq \bar{\mathbf{q}}^{\mathsf{t}}\bar{\mathbf{y}} + l - g(\bar{\mathbf{y}}). \tag{3}$$

If a vector $\bar{\mathbf{q}} \geq \mathbf{0}$ satisfies (1) for $\mathbf{y} \in \Delta$, then the cut (2) cannot remove any feasible solution of the original problem.

Proof. If $\bar{\mathbf{q}} \geq \mathbf{0}$ satisfies the inequality (1) in Δ, then $g(\mathbf{y}) < l$ everywhere in the interior of Δ, so the cut cannot remove any feasible solution there. We also have $g(\mathbf{y}) < l$ everywhere in the half-space where (2) is not satisfied, because g is nondecreasing in that half-space and every point $\mathbf{y}$ in that half-space satisfies $\mathbf{y} \leq \mathbf{y}_1$ for some point $\mathbf{y}_1 \in \Delta$. □

The area Δ is the slice of space between the hyperplane that determines the cut (2) and a parallel hyperplane that passes through the point $\bar{\mathbf{y}}$. We call a vector $\bar{\mathbf{q}} \geq \mathbf{0}$ that satisfies (1) for $\mathbf{y} \in \Delta$ a Δ-*subgradient* at $\bar{\mathbf{y}}$.

Cuts of the form (2) can be added for all violated constraints simultaneously in a single step, or for only some of the violated constraints. The next issues are: how do we obtain a subgradient, how can we make sure that g is concave, and what do we do if it is not concave? They are all addressed by heuristics.

3.3 Difficulties and Heuristic Cures

Getting a subgradient. To obtain a (tentative) Δ-subgradient $\bar{\mathbf{q}}$ at $\bar{\mathbf{y}}$, we simply use forward finite differences as follows. For $j = 1, \ldots, IP$, choose an integer $d_j \geq 0$. Compute

the function g at $\bar{\mathbf{y}}$ and at $\bar{\mathbf{y}} + d_j \mathbf{e}_j$ for $j = 1, \ldots, IP$, where $\mathbf{e}_j$ is the jth unit vector, with a 1 in position j and 0's elsewhere. We define $\bar{\mathbf{q}}$ as the IP-dimensional vector whose jth component is

$$\bar{q}_j = [g(\bar{\mathbf{y}} + d_j \mathbf{e}_j) - g(\bar{\mathbf{y}})]/d_j, \tag{4}$$

for all j. If we are sure that g is concave, taking $d_j = 1$ for all j is certainly the most natural choice, but the more general form with $d_j \geq 1$ is sometimes convenient, e.g., when the sample function is not smooth enough and/or not concave (we will return to this). If g is non-decreasing (which we assume), then $\bar{\mathbf{q}} \geq 0$. If g is concave and $d_j = 1$, then $\bar{q}_j$ is a subgradient for g viewed as a function of y_j alone (a one-dimensional function with discrete domain), for each j. Unfortunately, even under these conditions, $\bar{\mathbf{q}}$ is *not necessarily* a Δ-subgradient of g, because (1) may fail to hold in some diagonal direction (i.e., for some vector $\mathbf{y} - \bar{\mathbf{y}}$ having several positive components).

As an illustration, a two-dimensional function g may satisfy $g(0,0) = g(1,0) = g(0,1) = a$ and $g(1,1) > a$ and yet be concave. For instance, $g(y_1, y_2)$ can be defined by two planes that match these values, one plane for $y_1 \geq y_2$ and another plane for $y_1 \leq y_2$. In this case, we would get $\bar{\mathbf{q}} = (0,0)$ and this vector would not be a subgradient. The corresponding cut may then remove solutions $\mathbf{y}$ that do not satisfy (1) and do satisfy the service-level constraints of the original problem. So we can only view $\bar{\mathbf{q}}$ as a *heuristic guess* for a subgradient. On the other hand, in our experiments we routinely checked (1) in some positive diagonal directions, e.g., with each component of $\mathbf{y} - \bar{\mathbf{y}}$ equal to 1 or 2, and never found any serious problem in the cases where g was concave with respect to each of its coordinates separately.

Suppose that g satisfies the *submodularity* condition:

$$g(\mathbf{y} + \mathbf{d}_1 + \mathbf{d}_2) + g(\mathbf{y}) \leq g(\mathbf{y} + \mathbf{d}_1) + g(\mathbf{y} + \mathbf{d}_2) \tag{5}$$

for all non-negative integer vectors $\mathbf{d}_1$ and $\mathbf{d}_2$. For the functions $g_\bullet$ in our setting, this condition means that adding a vector $\mathbf{d}_2$ of agents to the center brings less gain if the current (vector) number is $\mathbf{y} + \mathbf{d}_1$ than if it is only $\mathbf{y}$ (i.e., the larger the current staffing, the smaller the marginal gain of adding more agents). It is quite reasonable to expect this to be true, at least when $\mathbf{y}$ is large enough. Under Condition (5), g is necessarily concave and the vector $\bar{\mathbf{q}}$ defined by (4) with $d_j = 1$ for all j always satisfies (1) for $\mathbf{y} \geq \bar{\mathbf{y}}$. However, it is not necessarily a Δ-subgradient because (1) may fail to hold for some vectors $\mathbf{y}$ such that $\bar{\mathbf{y}} - \mathbf{y}$ has both positive and negative components.

In general, if g is not concave or if $\bar{\mathbf{q}}$ is not a subgradient for some reason, the cut (2) may remove part of the set of feasible solutions (in terms of service-level) of the original problem. If this includes the optimal solution(s), we will simply end up with a suboptimal solution. In practice, this is not really catastrophic, because if we are not convinced that the solution is satisfactory, we can still run the algorithm again with different streams of random numbers and/or slightly different parameters.

Checking concavity. A second (related) issue concerns the concavity of g. Typically, g is a *convex* function of each coordinate of $\mathbf{y}$ when these coordinates are small, and a *concave* function when the coordinates are large. The rationale is that if there are much too few agents, almost no call is answered with the time limit and adding one agent is not likely to change this by much (at least for a large center). But if we keep adding agents, at some point the fraction $g(\mathbf{y})$ increases at a faster rate. Eventually, when there are enough agents so that most calls are answered within the time limit, adding more agents has little effect on $g(\mathbf{y})$. Figure 2 gives a one-dimensional illustration of this. The function $g(y)$ has an S-shape. It is convex for small values of y and concave for larger values. For the special case of an $M/M/s$ queue with arrival rate λ and service rate μ, the concavity of the service-level as a function of s was proved for $s > \lambda/\mu$ by Jagers and van Doorn (1991).

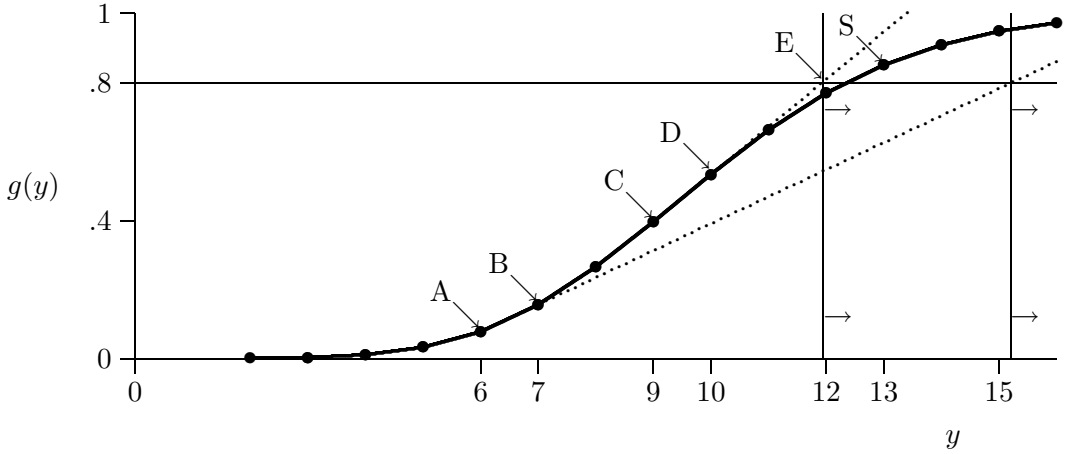

Figure 2: The S-shaped service-level function

In Figure 2, suppose that $l = 0.8$ and that we use $d_j = 1$ to get our subgradients. The optimal solution in this one-dimensional case is the smallest y for which $g(y) \geq 0.8$, namely $y = 13$. It corresponds to point S in the figure. If $\bar{y} = 9$, where g is concave, the subgradient $\bar{q}$ is the slope of the line that passes through points C and D in the figure. This line intersects the horizontal line $g(y) = l = 0.8$ at $y = 11.95$ (point E in the figure). The corresponding cut to be added, $qy \geq q\bar{y} + l - g(9)$, can be written as $y \geq 11.95$, or $y \geq 12$ because y must be integer. This constraint removes only infeasible solutions, i.e., values of y for which $g(y) < 0.8$. On the other hand, if $\bar{y} = 6$, we see that g is not concave in that area. The subgradient $\bar{q}$ at $\bar{y} = 6$ is the slope of the line going through points A and B (the lowest dotted line), which intersects the line $g(y) = 0.8$ at $y = 15.20$. The cut generation procedure would then add the constraint $y \geq 15.20$, or $y \geq 16$, which eliminates a large chunk of the feasible set, including the optimal solution.

Proving the joint concavity around the current solution appears much too difficult and impractical, so we only check the univariate concavity of g with respect to each coordinate y_j separately. More precisely, we check if $\bar{q}_j$ is a nondecreasing function of d_j, by verifying if

$$g(\bar{\mathbf{y}} + (d_j + 1)\mathbf{e}_j) - g(\bar{\mathbf{y}}) \leq \bar{q}_j(d_j + 1)$$

for each j. These are only *necessary* concavity conditions.

These (imperfect) concavity checks are expensive, because they require several additional simulations (at $\bar{\mathbf{y}} + (d_j + 1)\mathbf{e}_j$). One possibility is to simply bypass them altogether, to save time, at the risk of cutting out too much of the feasibility check. To reduce that risk, it is a good idea to take larger values of d_j when the current service-level is much smaller than its minimal value specified by the constraints. In our computational experiments, we found this to be more efficient than checking concavity, in the sense that it usually yields a very good solution in roughly half the time. For the experiments reported in Section 4, we took $d_j = 3$ when the service-level corresponding to the considered cut was less than 0.5, $d_j = 2$ when it was between 0.5 and 0.65, and $d_j = 1$ when it was greater than 0.65.

Initial constraints and what to do when we get into areas of non-concavity? At the beginning of the algorithm, if we relax all the nonlinear service-level constraints, the optimal solution of the relaxed problem is $\mathbf{y} = \mathbf{0}$. The functions g are obviously non-concave at that point, so we are in trouble right from the start! We must find a way of eliminating areas of non-concavity by adding other types of constraints to restrict the set of admissible solutions a priori, *before* generating any cut based on a subgradient. That is, impose $\mathbf{y} \in \mathcal{Y}$ for some set $\mathcal{Y}$ in which the functions $g_\bullet$ are more likely to be concave.

We obtain such constraints as follows. For any given period, we impose that the skill supply of agents during that period can cover a fraction α_k of the load of call type k during that period for all k, for some constants α_k which are usually close (or equal) to 1. To explain what this means, we assume for simplicity that call type k has arrival rate λ_k and service rate μ_k (independent of the agent group) in a given period, and that the number of agents of each group for that period is given by the vector $\mathbf{y} = (y_1, \ldots, y_I)^\mathtt{t}$. The *load* for call type k is $\rho_k = \lambda_k/\mu_k$.

Consider the graph of Figure 3, where there is an arc of infinite capacity from call type k to agent group i if and only if this agent group can handle call type k. Each arc going from an agent group i to node B on the right has a capacity equal to the number of agents in that group. Each arc going from node A to a call type k on the left has capacity equal to $\alpha_k\rho_k$, the load that we want to cover for that call type. We compute the maximal flow that can go from A to B in that graph. If we view the load of each call type as fluid, the flow that goes through call type k and agent group i corresponds to the load of call type k handled by agent group i, i.e., the number of agents of group i required to handle that load if these agents are busy 100% of their time. This is a real number (not necessarily an integer) and we call it the *skill supply* devoted to call type k. If the maximal flow equals

$$\bar{\rho} \stackrel{\text{def}}{=} \sum_{k=1}^{K} \alpha_k\rho_k,$$

this means that there is enough skill supply to handle the desired load for each call type. Otherwise, this is not possible. This is summarized in the next proposition.

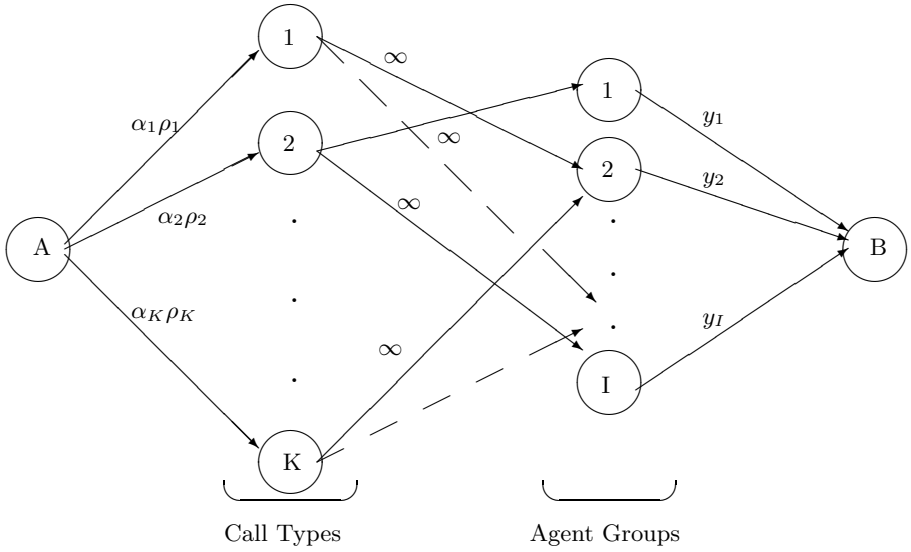

Figure 3: Covering load by skill supply

Proposition 4 *The maximum flow in the graph of Figure 3 is $\bar{\rho}$ if and only if there is enough skill supply in the agent groups to cover a load of $\alpha_k\rho_k$ for call type k, simultaneously for all k (for a total load of $\bar{\rho}$).*

For any given solution $\mathbf{y} = \bar{\mathbf{y}}$, the maximum flow in the graph can be computed easily by a standard max-flow algorithm. If the flow is less than $\bar{\rho}$, the algorithm finds a minimum cut that provides a valid linear inequality on $\mathbf{y}$ needed to reach the required maximum flow, but violated by the current solution. We add this constraint and find a new solution $\bar{\mathbf{y}}$. This is repeated until the maximum flow reaches $\bar{\rho}$.

This procedure is performed at the beginning of the algorithm and is repeated each time we obtain a new solution $\bar{\mathbf{y}}$ after having added linear inequalities based on service-level constraints. The computational requirement for solving this max-flow problem on a sparse bipartite network such as ours is almost insignificant in comparison with the overall computing time, regardless of the values of K and I (which may reach 100 in large call centers). In our computational experiments, we took all α_k's equal to 1, except for a situation that we will describe in a moment.

It is important to underline that this procedure is a heuristic, in the sense that being able to handle the load (or a given fraction of it, as specified above) is *neither necessary nor sufficient* for the model to be stable. It is not necessary because abandonments can keep the system stable if the load cannot be covered. And even without abandonments, if the load is not covered during a given period, a queue will build up during that period but these calls could be handled during the next period, in which there could be extra

capacity. These are the reasons why we may not always take $\alpha_k = 1$ for all k. In the case of steady-state models without abandonments, however, we should take $\alpha_k \geq 1$. But even if the maximum flow exceeds $\bar{\rho}$ with $\alpha_k > 1$ for all k, i.e., if there is enough skill supply to simultaneously cover more than the load for each call type, this does not necessarily imply that the system is stable (see Example 1 in Section 4, and Garnett and Mandelbaum, 2000, for counterexamples). Stability in this case generally depends on the priority rules in the routing strategies. The procedure is nevertheless very useful to cut out areas of non-concavity of the functions $g_\bullet$.

Even when the skill supply can cover the load of all call types, it may happen that certain call types have extremely low service-level if the routing rules give them low priority. When we want to generate a subgradient cut based on the service-level threshold for such a call type, we run into a non-concavity problem. The "subgradient" is often nearly flat (zero or almost zero) and the corresponding cut would remove all the good solutions. In this case (which typically happens for highly unbalanced systems, where certain types of calls have very low priority but yet require a minimal service-level), we need a different type of heuristic. What we do is simply increase the value of α_k used in the max-flow problem for the call type k having the highest service-level gap, i.e., the smallest value of $g_{k,p}(\bar{\mathbf{y}}) - l_{k,p}$ or $g_k(\bar{\mathbf{y}}) - l_k$. The trick is to increase α_k just enough to get out of the non-concavity area. An appropriate value could be found by trial and error, checking the concavity and the service-levels obtained for each trial. In our computational experiments, we adopted a heuristic that increases α_k so that the resultant service-level for call type k is between 0.01 and 0.1. For the examples that we tried, this appeared always sufficient to get rid of these flat subgradients, and taking a larger α_k was counterproductive (empirically).

When very low service-levels are obtained for certain types of calls even if the global service-level is high, this may indicate poor routing strategies. In that case, significant improvements on the service-level per call type can often be obtained by changing the routing rules, which is likely to be less expensive than adding more agents.

Cut generation priorities. The problem of certain call types having zero (or near zero) service level happens more frequently when the global service-level is still low as well. Moreover, when the global service-level is higher, the service-level for a given type of call tends to be concave over a wider range of values than when the global service-level is low. We observed empirically that in many cases, when the global service level is high, the service-level of a given call type is concave as a function of a given y_j practically as soon as it becomes positive.

For this reason, as long as the global service-level is below a given threshold ℓ_*, we only generate cuts based on the global service-level constraint. After that threshold has been reached, at each step of the algorithm we generate and add a cut for each service-level constraint that is not satisfied. All these constraints are added simultaneously. Adding several constraints at the same time generally makes the algorithm more efficient, because it reduces the total number of steps and thus the number of simulations runs. On the other hand, adding bad cuts (obtained in non-concave areas) can be damaging. So there is a compromise to be made and the choice of ℓ_* decides on that compromise. In our experiments, that threshold was 0.65.

Problems with the smoothness of sample functions. Yet another difficulty comes from the fact that the sample functions $G_{n,\bullet}$ are typically less smooth than their "expected value" counterparts $g_\bullet$. Because of that, $G_{n,\bullet}$ is often locally non-concave in the area where $g_\bullet$ is concave. The explanation is that for a given sample path, if we increase the number of agents by one at a time, the actual number of calls that get served within the time limit may increase sometimes by a large amount, sometimes by a small amount, sometimes not at all, and the *amount of increase* is not necessarily monotone, for a small n, even if it becomes monotone when $n \to \infty$. When we increase the sample size n they become smoother, so the problem diminishes, but it may take a very large value of n before it disappears completely, and that value is random. Increasing n also makes the simulation much more expensive.

This is illustrated in Figure 4, which shows the actual sample function $G_{n,k}$ for a given call type k as a function of the number of agents of a given group that can handle that call type, with the number of agents in all other groups held fixed, in a simulation of 50 hours of operation of the large example of Section 4. The sample function is flat between 6 and 7, between 22 and 23, and between 24 and 25. If we use $d_j = 1$ in this case, at $\bar{y} = 6$ and 22, (points A and B in the figure), the subgradient is zero and the corresponding cut removes everything! The subgradient is also zero at $\bar{y} = 24$ (C in the figure) but the constraint is already satisfied at that point. At D and E, the subgradient is not zero, but it is small enough so the corresponding cut would impose an excessive lower bound on y.

This type of problem can be detected by univariate concavity checks. When it occurs, we could simply use larger values of d_j in the subgradient estimators. In the example on the figure, for instance, taking $d_j = 2$ or 3 would do fine. A sensible heuristic could be to try $d_j = 1$, 2, and 3, and take the one that gives the largest slope. When the service-level $G_{n,\bullet}$ at the current solution is still much smaller than the threshold, it is usually a good idea to start directly with some $d_j > 1$. In our computational experiments of Section 4 we simply bypassed all concavity checks (to save time) and directly used larger values of d_j as explained earlier.

With all these heuristics, subgradients close to zero are still encountered once in a while and we must avoid adding the corresponding cuts. For that, we use another threshold: when all coordinates of $\bar{\mathbf{q}}$ are smaller in absolute value than this threshold, we do not add this cut. The other cuts added at the same step will usually take care of improving the situation enough to circumvent the problem. This threshold was set at 0.01 in our experiments.

Relaxing integrality and rounding up. For problem instances where the vector $\mathbf{y}$ has a small dimension, we can solve the sample problem at each step (i.e., after adding new constraints) by integer programming (IP). The simulation time (to evaluate the service-levels required to compute $\mathbf{q}$) typically dominates the total CPU time in that case. But when the problem becomes too large, the time required for solving the integer program becomes excessive (this time increases exponentially with the size of the problem) and we must find an alternative.

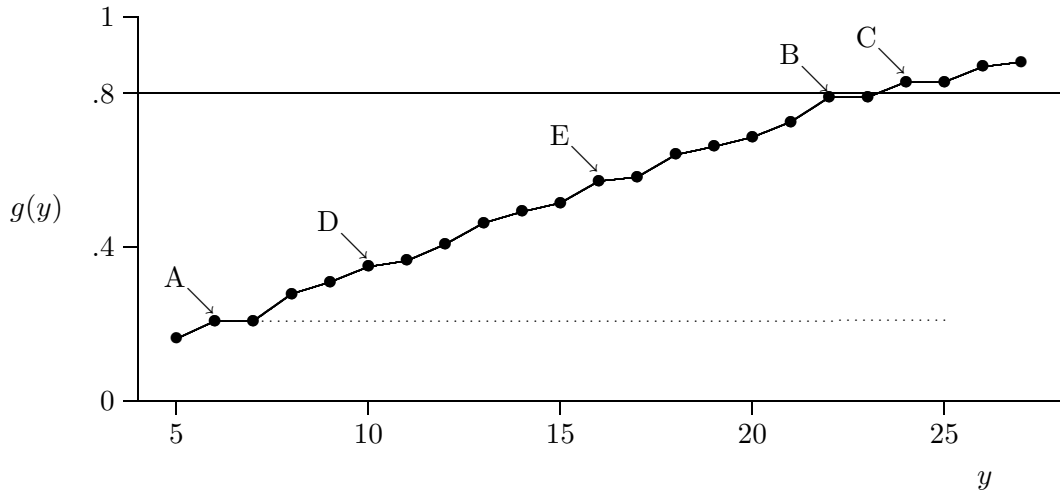

Figure 4: Sample service-level function of a call type (from large example) when all except one agent groups are fixed and the global service-level is greater than 0.65.

What we do in that case is simply relax the integrality constraints: At each iteration, we solve the standard linear programming (LP) problem instead of the IP problem. Then, we round up each component of the solution to an integer. Obviously, all constraints satisfied by the optimal LP solution are also satisfied by the rounded up vector. We then run simulations with this rounded up solution to check the service-level constraints, check concavity, and estimate a subgradient.

At the last iteration, when we find a rounded up solution that satisfies all service-level constraints, we perform a local search around that solution to see if it can be easily improved. For the numerical experiments reported here, the local search was rather simplistic: we just decreased each component of the vector $\bar{\mathbf{y}}$ by 1, by decreasing order of agent cost, until we reached infeasibility. This local search procedure could certainly be improved.

We also experimented this approach on medium-sized problems, where IP was practical but yet took a significant fraction of the total CPU time. With the LP+rounding approach, the time for solving the LP was almost negligible compared with the simulation time (so the overall algorithm was faster) and the results were generally with 1% of the IP results.

4 Computational Experiments

The numerical examples considered here are for a single period only and we solve the staffing problem under the assumption that the system is in steady-state. The arrivals are assumed to be from a stationary Poisson process with rate λ_k for each k and the service times are exponential with rate μ_k for each k.

For all experimental results reported here, the thresholds and parameters of the heuristics were selected as mentioned in the previous section. Thus, all examples were solved with the same algorithm. Our computer had a 2.0GHz AMD Athlon 64 Processor 3200+ running Linux, SUN Java Development Toolkit 1.4.2, and CPLEX 8.1 was our LP/IP solver. The simulations were performed using a contact center simulation package currently developed in our laboratory by Éric Buist, in the Java language, under the framework of the SSJ simulation package (L'Ecuyer, 2004). In our simulations, we always start the system full (all agents busy but no call in the queues). At each value of $\mathbf{y}$ we perform a single simulation run of length $T + T/20$ (in the simulation time frame) for some constant T, divide it into 21 batches of length $T/20$, and discard the first batch. The 20 other batches are used to compute estimates and confidence intervals for the service-levels, globally and for each call type. Unless stated otherwise, $T = 50$ hours, so we have 2.5 hours of warmup and 50 hours of simulation. For the smaller examples, we tried $T = 500$ hours for comparison. The results differed by less than 1% except for a single case where the longer simulation found a much better solution (see Example 1 below). The total computing times are approximately proportional to T.

4.1 Staffing a Center of Moderate Size

We first consider two examples of moderate size, one with 5 call types and 12 agent groups, the other with 20 call types and 15 agent groups. Example 1 is adapted from Koole and Talim (2000).

For the routing, we assume that each call type has an ordered list of agent groups, used to select an available agent upon arrival. If all agent groups in that list are busy, the call joins a queue (one FIFO queue per call type). Each agent group has an ordered list of call type queues. When an agent from this group becomes available, it selects the first non-empty queue from its list and picks the first call in the queue.

Example 1 In our first example, there are 5 call types and 12 skill groups, defined by the lines and columns of Table 1, respectively. The priorities are very simple: An arriving call checks for an available agent by numeric order of agent groups and an agent that becomes available also looks at call queues in numeric order. Such simplistic priorities make the system highly unbalanced and the problem is then more difficult to solve because the service-level of the low-priority call types tends to be very low (in the non-concavity zone).

k	agent group i											
1	1		3	4	5		7	8	9		11	12
2			3			6	7	8			11	12
3		2		4		6	7		9	10	11	12
4					5					10		12
5								8	9	10	11	12

Table 1: Skill groups for the first example

The cost of an agent with $s+1$ skills is assumed to be $1+\kappa s$ where $\kappa = 0.10$. The arrival rates and service rates per hour are $\lambda_1 = \lambda_3 = \lambda_5 = 440$, $\lambda_2 = \lambda_4 = 540$, and $\mu_k = 12$ for all k. We require the overall service-level to be at least $l = 0.8$. For the values of l_k, we consider two cases: $l_k = 0$ (no constraint on the service-level per call) and $l_k = 0.5$. Regarding the abandonments we also consider two cases: (1) no abandonment and (2) abandonment rate of 10 (per hour) for each call type. This gives a total of four combinations. For each of them, we use the original method that solves an IP at each step and the method that replaces this IP by an LP and round up every solution. The results for the eight cases are in Table 2. For each case, the table indicates if there were abandonments (aban.), gives the minimal service-level per call type ($l_k = 0.0$ or 0.5), says if the IP or the LP was solved at each step (algo.), then gives the total number of cuts generated by the algorithm (cuts), the total CPU time in seconds (CPU), the global service-level for the final solution (QoS), the global service-level estimated by the Koole-Talim approximation (Koole et al., 2003) for that same solution (QoS KT), the service-level per call type (QoS per call type) for the final solution, and the staffing vector $\mathbf{y}$ in the final solution. The service-level values reported here are accurate to approximately $\pm 0.02\%$ with 95% confidence. The Koole-Talim approximation can be applied only when there is no abandonment.

Except for one case (see below), solving the IP as an LP at each step (i.e., using the rounding method) increases the value of the objective function for the final solution, by about 1% on average, while reducing the CPU time roughly by a factor of 2 or 3. For a small example such as this one, it may be worth spending this additional CPU time to get a better solution, i.e., solve the IP at each iteration. For larger examples, however, the factor of CPU times between the IP and LP cases can be huge, and solving the LP eventually becomes the only practical solution. The exception is the case where $l_k = 0.5$ and there is no abandonment. Then, the IP and LP methods give totally different staffing vectors, with a large difference in the objective function value, and the LP does better than the IP. For this particular (difficult) case, with 500 hours of simulation, the IP method found a much better solution, with value of 244.3, whereas LP found one with value 248.2.

Another interesting observation is that when $l_k = 0$, call type 5 has very poor service-level in the optimal solution: only a small fraction are answered within 20 seconds when the are abandonments and none when there are no abandonments. In the no-abandonment case, the system is actually unstable for call type 5: An infinite queue does build up with the retained solution, even if there is enough skill supply to cover more than the load of each call type simultaneously. (This explains the fact that the average of service-levels per call type weighted by their arrival rates does not match the global service-level; several calls of type 5 are still in the queue at the end of the simulation and are not accounted for). The unstability is due to the fact that each agent group that can handle calls of type 5 consider them with the lowest possible priority, and can handle at least two other types of calls with higher priorities. With these fixed (and unbalanced) priority rules, when we add agents to these groups, these agents first take care of other call types before handling

aban.	l_k	algo.	cuts	CPU (sec)	obj.	QoS	QoS KT
QoS per call type							
staffing vector							
no	0.0	IP	9	116	220.9	.804	.115
(.99, .90, .99, .80, .00)							
(4 , 36 , 5 , 28 , 45 , 45 , 1 , 13 , 0 , 24 , 0 , 0)							
no	0.0	LP	6	66	222.1	.803	.454
(.99, .88, .99, .84, .00)							
(36 , 38 , 0 , 0 , 45 , 46 , 2 , 15 , 0 , 23 , 0 , 0)							
yes	0.0	IP	10	114	219.5	.801	
(.99, .93, .95, .84, .21)							
(32 , 30 , 0 , 1 , 48 , 44 , 0 , 30 , 0 , 16 , 0 , 0)							
yes	0.0	LP	4	45	220.5	.804	
(.99, .94, .97, .88, .12)							
(36 , 30 , 15 , 0 , 45 , 39 , 0 , 17 , 0 , 21 , 0 , 0)							
no	0.5	IP	9	67	263.2	.863	.860
(.99, .88, .99, .92, .59)							
(0 , 30 , 0 , 24 , 0 , 0 , 0 , 0 , 0 , 65 , 0 , 92)							
no	0.5	LP	7	50	251.9	.803	.866
(.96, .51, 1.00, .94, .62)							
(0 ,47 , 0 , 13 , 0 , 14 , 0 , 66 , 0 , 80 , 0 , 0)							
yes	0.5	IP	35	280	224.7	.801	
(.99, .61, .99, .85, .57)							
(26 , 25 , 11 , 1 , 36 , 39 , 0 , 0 , 30 , 35 , 0 , 0)							
yes	0.5	LP	14	131	225.4	.809	
(.99, .76, .98, .76, .54)							
(25 , 24 , 0 , 7 , 45 , 44 , 0 , 12 , 33 , 14 , 0 , 0)							

Table 2: Results for Example 1

calls of type 5 and there is not enough capacity left to handle their load. So we must add a large number of these (rather generalist) agents to really increase the service-level of this call type. This is why the objective function increases significantly when we impose a minimal service-level per call type with $l_k = 0.5$. This increase is much more pronounced when there are no abandonments, because the abandonments (of all call types) during the periods of high congestion has a large impact in improving the service-level of call type 5. For the case where $l_k = 0.5$ with abandonments, with the retained solutions with both IP and LP, the abandonment ratio is approximately 3.5% globally and 12% for call type 5. Of course, less expensive solutions could be found if we were allowed to change the priority rules of the agents.

When solving the case $l_k = 0.5$ without abandonment, we first add cuts only for the global constraint (as described in Section 3.3) and come up with a solution with a service-level of zero for call type 5. At that point, we cannot add a subgradient cut based on the service-level constraint of this call type, because the subgradient is zero. This is a case where we must inflate α_5 in the max-flow problem.

The Koole-Talim approximation for the global service-level gives totally unreliable values when the system is highly unbalanced and some call types have very low service-levels. It behaves much better for balanced systems. We also observed this in other examples.

k	agent group i													
1	1		3		5		7		9					
2	1									11		13		15
3		2		4		6		8		10				
4				4							12		14	
5	1	2			5					11				
6			3				7	8		10				
7					5				9		12	13		
8					5	6				10		12		14	15
9		2		4	5	6				10				
10					5	6			9				13	14	
11	1				5			8		10		12			
12				4					9		11			14	15
13		2			5		7			10					15
14			3					8	9				13		15
15		2				6	7							14	
16	1				5					10		12			
17		2				6					11				
18			3	4									13	14	
19		2						8				12			15
20			3			6		8				12	13	14	

Table 3: Skill groups for the second example

Example 2 This is a slightly larger example, with 20 call types and 15 agent groups, defined by the lines and columns of Table 3. The priority rules, cost function, abandonment rates, and service-level requirements, are as in the first example. The arrival rates and service rates per hour are $\lambda_1 = \lambda_2 = 240$, $\lambda_3 = \lambda_{17} = 160$, $\lambda_4 = \lambda_7 = \lambda_9 = \lambda_{14} = \lambda_{19} = \lambda_{20} = 260$, $\lambda_5 = \lambda_8 = \lambda_{16} = \lambda_{18} = 130$, $\lambda_6 = \lambda_{15} = 230$, $\lambda_{10} = 125$, $\lambda_{11} = 235$, $\lambda_{12} = 155$, $\lambda_{13} = 225$, and $\mu_k = 12$ for all k. The results are in Table 4.

Again, the rounding method increases the objective function value by about 1% on average, and reduces the CPU time by a factor of 1 or 4. The unbalanced priority rules bring the same types of problems as in Example 1. The low priority calls get extremely poor service-level with $l_k = 0$ and the objective function value increases significantly when we impose $l_k = 0.5$ and there are no abandonments. For $l_k = 0.5$ with abandonments, for the retained solutions with IP and LP, the abandonment ratio is approximately 3.5% globally and higher for the low-priority call types. For instance, for call types 16, 18, and 20, it is 9%, 10%, and 14%, respectively.

aban.	l_k	algo.	cuts	CPU (sec)	obj.	QoS	QoS KT
QoS per call type							
staffing vector							
no	0.0	IP	10	232	467.7	.802	.079
(1.00, .99, .99, .93, .98, .99, .76, .00, .61, .30, .82, .97, .98, .94, .94, .27, .88, .89, .00, .46)							
(31 , 0 , 55 , 61 , 0 , 0 , 74 , 21 , 33 , 0 , 54 , 12 , 0 , 0 , 0)							
no	0.0	LP	2	62	472.6	.818	.280
(1.00, .99, .99, .96, .99, .99, .94, .00, .76, .77, .86, .99, .98, .90, .93, .37, .97, .81, .00, .00)							
(30 , 0 , 27 , 68 , 0 , 0 , 78 , 22 , 50 , 0 , 57 , 11 , 0 , 0 , 1)							
yes	0.0	IP	7	184	466.6	.800	
(1.00, 1.00, .99, .95, .99, .99, .93, .34, .82, .80, .85, .99, .98, .91, .94, .48, .96, .85, .03, .07)							
(32 , 0 , 23 , 72 , 0 , 0 , 77 , 17 , 52 , 0 , 51 , 0 , 0 , 1 , 15)							
yes	0.0	LP	7	164	470.8	.807	
(1.00, .99, .99, .95, .99, .99, .84, .15, .83, .63, .87, .99, .99, .94, .96, .45, .95, .92, .03, .39)							
(30 , 0 , 38 , 72 , 0 , 0 , 77 , 22 , 43 , 0 , 50 , 0 , 0 , 0 , 11)							
no	0.5	IP	40	486	501.6	.872	.799
(1.00, .98, .99, .99, .96, .99, .94, .50, .91, .78, .98, .96, .86, .98, .57, .75, .56, .97, .69, .78)							
(46 , 9 , 52 , 63 , 0 , 0 , 42 , 51 , 42 , 0 , 27 , 13 , 0 , 10 , 0)							
no	0.5	LP	26	226	500.4	.868	.805
(1.00, .99, .99, .96, .97, .99, .98, .50, .64, .90, .98, .98, .90, .98, .61, .76, .79, .90, .71, .70)							
(41 , 0 , 41 , 55 , 0 , 0 , 51 , 53 , 56 , 0 , 36 , 16 , 0 , 0 , 8)							
yes	0.5	IP	9	105	474.0	.831	
(1.00, .99, .99, .90, .97, .99, .76, .57, .62, .50, .98, .95, .98, .95, .89, .55, .78, .52, .76, .51)							
(29 , 0 , 9 , 59 , 0 , 0 , 69 , 72 , 40 , 0 , 41 , 0 , 0 , 0 , 20)							
yes	0.5	LP	12	87	473.4	.825	
(1.00, 1.00, .97, .83, .99, .99, .79, .91, .50, .53, .77, .99, .99, .97, .94, .53, .88, .88, .50, .51)							
(43 , 0 , 45 , 61 , 0 , 0 , 69 , 0 , 40 , 0 , 42 , 0 , 0 , 0 , 43)							

Table 4: Results for Example 2

4.2 Staffing a Large Center

We now consider a model strongly inspired by a large real-life call center operated by Bell Canada. There are 89 call types and 65 skill groups. We do not provide the details of the arrival, service, and abandon rates, and the call push/pull matrices, because of space limitation and proprietary nature of some of these details. In summary, the model assumes that the arrivals are Poisson with rates λ_k varying from 1.046 to 416.6, and the global rate is 3581.7. The services times are assumed exponential with rates μ_k varying from 0.6777 to 600. The *aggregate load* ($\sum_k \lambda_k/\mu_k$) is 500. The patience times are exponential, with an abandonment rate of 2.0 for all call types (all these numbers are per hour). For the service-level constraints, we take $l = 0.8$ globally and $l_k = 0.5$ for each call type.

Both the call types and the agent groups are split in two locations. One location has 22 call types and 15 agent groups; the second one has 43 call types and 74 agent groups. The number of skills per agent group goes from 1 to 24. We model the cost of an agent with $s + 1$ skills as $1 + \kappa s$ where $\kappa = 0.05$.

The router uses a set of priority rules named "local specialist routing policy," which tries to assign any incoming call primarily to an agent based in the location from where the

call originates. If more than one such local agent can handle this call, the router chooses the agent with the smallest number of skills. In case of equalities, the agent with the longest idle time is chosen. If no local agent is available to serve the call, it is put in a queue corresponding to its type. When a call has spent more than 6 seconds in the queue, the router tries to assign it to any agent from the other location. If this succeeds, the call leaves the queue and is served remotely. On the other hand, when an agent becomes free, all the local queues are queried for a call and the call with the smallest waiting time is taken. By using this policy, the router behaves as there was only one queue for all call types. If there are still free agents after the queues are queried, the query is made a second time, allowing agents to serve calls from the other location.

We first tried to solve this problem as an IP at each step, but the CPU time requirement was unreasonably high. So we solved the LP (with rounding up) instead. The resolution took 37 minutes of CPU time. We obtained a solution with 526 agents (for a load of 500), for which the objective function value was 669.35. The average number of skill per agent is 6.45, and the retained staffing vector has 37 nonzero entries ranging from 1 to 68. For that solution, the 50-hour simulation (used by our algorithm) estimates global service-level at 0.83 ± 0.02 (with 95% confidence), the average agent occupancy at 94.5%, and the abandonment ratio at 1.1%. We also ran a 5000-hour simulation with this retained solution to check the service-levels with better precision. This time, the global service level was estimated as 0.845 ± 0.002, and three call types had estimated service-levels slightly below 0.5, namely 0.49, 0.47, and 0.47. After observing this, it would be easy to add a few more cuts to increase these three service-levels and do one or two additional iterations, but we did not do that.

Interestingly, the five call types having the largest arrival rates, which account for 42.3% of the total call volume, are also those that have the largest service-level, all around 0.99, in the retained solution. The explanation is that these call types are favored by the current (fixed) priority rules. In contrast, the sixth most frequent call type has its service-level at the lower bound of 0.5, with an abandonment ratio of 5.2%. This call type can be handled by a single agent group. This group has 53 agents with 11 different skills and is at a different location than the call type, which explains the poor service-level. Its occupation rate is 93.7%, a little below the average.

In this model, the objective function could be reduced significantly by reducing the average number of skills per agent and optimizing the skill mixes, agent group locations, and routing rules and priorities. This is beyond the scope of this paper and will be addressed in future work.

Appendix

This appendix is not meant to be published in a journal's version of the paper. It provides additional detailed information by giving versions of Tables 2 and 4 for experiments where the system was simulated for 500 hours of operation instead of 50 hours in the sample problem, thus giving more precision in the service-levels (by a factor of $\sqrt{10}$).

aban.	l_k	algo.	cuts	CPU (sec)	obj.	QoS	QoS KT
QoS per call type							
staffing vector							
no	0.0	IP	6	388	218.0	.801	.115
(.99, .82, .97, .87, .00)							
(22 , 36 , 1 , 0 , 60 , 45 , 0 , 16 , 0 , 21 , 0 , 0)							
no	0.0	LP	4	229	221.2	.802	.386
(.99, .82, .97, .87, .00)							
(22 , 36 , 1 , 0 , 60 , 45 , 0 , 16 , 0 , 21 , 0 , 0)							
yes	0.0	IP	12	1173	217.5	.801	
(.99, .93, .98, .89, .11)							
(34 , 29 , 3 , 0 , 45 , 51 , 0 , 12 , 0 , 26 , 0 , 0)							
yes	0.0	LP	5	504	219.3	.802	
(.99, .89, .98, .91, .13)							
(31 , 36 , 0 , 0 , 51 , 46 , 0 , 17 , 0 , 21 , 0 , 0)							
no	0.5	IP	13	882	244.3	.801	.600
(.99, .81, .98, .69, .52)							
(11 , 8 , 0 , 0 , 2 , 1 , 0 , 97 , 0 , 88 , 0 , 0)							
no	0.5	LP	13	748	248.2	.809	.643
(.99, .86, .99, .59, .63)							
(1 , 6 , 0 , 0 , 0 , 0 , 0 , 102 , 23 , 76 , 0 , 0)							
yes	0.5	IP	39	2966	221.3	.801	
(.99, .96, .92, .60, .50)							
(21 , 27 , 27 , 0 , 44 , 20 , 0 , 44 , 7 , 10 , 0 , 0)							
yes	0.5	LP	8	713	224.0	.825	
(.99, .65, .99, .97, .50)							
(24 , 31 , 50 , 0 , 46 , 2 , 0 , 0 , 0 , 51 , 0 , 0)							

Table 5: Results for Example 1 with 500hrs of simulation

aban.	l_k	algo.	cuts	CPU (sec)	obj.	QoS	QoS KT
QoS per call type							
staffing vector							
no	0.0	IP	4	443	466.8	.813	.079
(1.00, .99, .99, .95, .99, .99, .93, .00, .77, .79, .86, .99, .97, .89, .91, .35, .96, .8103, .00, .00)							
(31 , 0 , 22 , 68 , 0 , 0 , 76 , 22 , 62 , 0 , 49 , 0 , 0 , 0 , 11)							
no	0.0	LP	3	171	467.7	.820	.079
(1.00, .99, .99, .96, .99, .99, .94, .00, .76, .77, .86, .99, .98, .90, .93, .37, .97, .81, .00, .00)							
(30 , 0 , 22 , 72 , 0 , 0 , 77 , 22 , 54 , 0 , 53 , 0 , 0 , 0 , 11)							
yes	0.0	IP	3	855	466.5	.800	
(1.00, .99, .99, .96, .99, .99, .94, .23, .83, .83, .87, .99, .98, .92, .94, .46, .97, .86, .02, .06)							
(31 , 0 , 22 , 73 , 0 , 0 , 77 , 20 , 53 , 0 , 51 , 0 , 0 , 0 , 13)							
yes	0.0	LP	3	657	469.3	.806	
(1.00, .99, .99, .95, .99, .99, .95, .59, .82, .84, .77, .99, .99, .93, .95, .35, .97, .86, .08, .07)							
(30 , 0 , 22 , 73 , 0 , 0 , 77 , 11 , 54 , 0 , 52 , 0 , 0 , 0 , 23)							
no	0.5	IP	40	2744	496.7	.863	.757
(1.00, .98, .99, .99, .92, .99, .94, .50, .87, .77, .97, .95, .86, .98, .57, .69, .54, .95, 69, .76)							
(40 , 0 , 49 , 64 , 0 , 0 , 49 , 54 , 39 , 0 , 31 , 19 , 2 , 5 , 1)							
no	0.5	LP	36	1926	505.3	.891	.837
(1.00, .99, .99, .96, .96, .99, .89, .50, .85, .80, .99, .96, .93, .99, .79, .73, .76, .96, .83, .88)							
(39 , 0 , 50 , 62 , 0 , 19 , 51 , 60 , 34 , 0 , 33 , 2 , 9 , 0 , 0)							
yes	0.5	IP	18	1729	472.4	.827	
(1.00, .99, .97, .84, .99, .99, .78, .91, .52, .51, .77, .99, .99, .97, .92, .51, .89, .89, .50, .55)							
(43 , 0 , 48 , 62 , 0 , 0 , 65 , 0 , 39 , 0 , 42 , 0 , 0 , 0 , 43)							
yes	0.5	LP	12	1316	474.6	.844	
(.99, .99, .99, .93, .96, .99, .82, .61, .69, .59, .98, .96, .97, .95, .89, .50, .78, .50, .78, .52)							
(27 , 0 , 2 , 63 , 0 , 0 , 69 , 75 , 42 , 0 , 41 , 0 , 0 , 0 , 21)							

Table 6: Results for Example 2 with 500hrs of simulation

References

Atlason, J., Epelman, M. A., and Henderson, S. G. (2004). Call center staffing with simulation and cutting plane methods. *Annals of Operations Research*, 127:333–358.

Avramidis, A. N., Deslauriers, A., and L'Ecuyer, P. (2004). Modeling daily arrivals to a telephone call center. *Management Science*, 50(7):896–908.

Ellis, R. S. (1985). *Entropy, Large Deviations, and Statistical Mechanics*. Springer Verlag.

Gans, N., Koole, G., and Mandelbaum, A. (2003). Telephone call centers: Tutorial, review, and research prospects. *Manufacturing and Service Operations Management*, 5:79–141.

Garnett, O. and Mandelbaum, A. (2000). An introduction to skill-based routing and its operational complexities. teaching notes, Technion, Israel.

Ingolfsson, A. and Cabral, E. (2002). Combining integer programming and the randomization method to schedule employees. Technical report, School of Business, University of Alberta, Edmonton, Alberta, Canada. Preprint.

Jagers, A. A. and van Doorn, E. A. (1991). Convexity of functions which are generalizations of the Erlang loss function and the Erlang delay function. *SIAM Review*, 33:281–282.

Jennings, O. B., Mandelbaum, A., Massey, W. A., and Whitt, W. (1996). Server staffing to meet time-varying demand. *Management Science*, 42(10):1383–1394.

Kelley Jr., J. E. (1960). The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712.

Koole, G. and Mandelbaum, A. (2002). Queueing models of call centers: An introduction. *Annals of Operations Research*, 113:41–59.

Koole, G., Pot, A., and Talim, J. (2003). Routing heuristics for multi-skill call centers. In *Proceedings of the 2003 Winter Simulation Conference*, pages 1813–1816. IEEE Press.

Koole, G. and Talim, J. (2000). Exponential approximation of multi-skill call center architecture. In *Proceedings of QNETs 2000*, pages 23/1–10, Ilkley (UK).

Law, A. M. and Kelton, W. D. (2000). *Simulation Modeling and Analysis*. McGraw-Hill, New York, third edition.

L'Ecuyer, P. (2004). *SSJ: A Java Library for Stochastic Simulation*. Software user's guide, Available at `http://www.iro.umontreal.ca/~lecuyer`.

L'Ecuyer, P., Simard, R., Chen, E. J., and Kelton, W. D. (2002). An object-oriented random-number package with many long streams and substreams. *Operations Research*, 50(6):1073–1075.

Shwartz, A. and Weiss, A. (1995). *Large Deviations for Performance Analysis*. Chapman and Hall, London.

Vogel, S. (1994). A stochastic approach to stability in stochastic programming. *Journal of Computational and Applied Mathematics*, 56:65–96.

Wallace, R. B. and Whitt, W. (2004). Resource pooling and staffing in call centers with skill-based routing. manuscript.

Yakowitz, S., L'Ecuyer, P., and Vázquez-Abad, F. (2000). Global stochastic optimization with low-discrepancy point sets. *Operations Research*, 48(6):939–950.