Staffing multi-skill call centers via search methods and a performance approximation

Athanassios N. Avramidis, Wyean Chan, and Pierre L'Ecuyer

Département d'Informatique et de Recherche Opérationnelle

Université de Montréal, C.P. 6128, Succ. Centre-Ville, Montréal, H3C 3J7, CANADA

Abstract. We address a multi-skill staffing problem in a call center where the agent skill sets are exogenous and the call routing policy has well-specified features of overflow between different agent types. Constraints are imposed on the service level for each call class, defined here as the steady-state fraction of calls served within a given time threshold, where calls that abandon after having waited for service less than the threshold are excluded. We develop an approximation of these service levels, allowing an arbitrary overflow mechanism and allowing customer abandonment. We then develop a two-stage heuristic that finds good solutions to mathematical programs with such constraints. The first stage uses search methods supported by the approximation. Because service-level approximation errors may be substantial, we adjust the solution in a second stage in which performance is estimated by simulation. We solve realistic problems of varying size and routing policy. Our approach is shown to be competitive with (and often better than) previously available methods.

[Supplementary materials are available for this article. Go to the publisher's online edition of *IIE Transactions* for the following free supplemental resource: Online Appendix.]

1 Introduction

Call centers usually handle several types of calls distinguished, for example, by the desired language of communication or the level of skill necessary for delivering technical support. It is usually not possible or cost-effective to train every agent to be able to handle every call class. Thus, frequently, one encounters a *multi-skill* call center, with various *call classes* and also various *agent types*, usually defined according to their *skill set*, i.e., the subset of call classes they can handle. *Skill-based routing* (SBR), or simply *routing*, refers to the rules that control the call-to-agent and agent-to-call

assignments. Most modern call centers perform skill-based routing (Koole and Mandelbaum, 2002).

Call center managers routinely impose constraints on the center's performance. A commonly encountered performance measure is the *service level* (SL), usually defined as the long-term fraction of calls whose waiting time is no larger than a given constant. Call center planners face the problems of determining appropriate staffing levels and agent work schedules. In a *staffing* problem, the day is divided into periods and one simply decides the number of agents of each type 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 type in each work schedule. This determines the staffing indirectly, while making sure that it corresponds to a feasible set of work schedules.

Insights on the coordination of skill set design, staffing, and routing decisions for multi-skill centers are offered by Wallace and Whitt (2005). First, endowing agents with two skills and employing a routing that balances agents' priorities over different call classes, they obtain service levels that are essentially as good as for a system where all agents have all skills. That is, if such a routing policy is practical, then training agents to have more than two skills adds little to performance. Second, assuming control over agent skill sets, staffing counts, and routing, they meet nearly exactly (i.e., do not exceed) target service levels set for each call class.

We consider a single-period staffing problem where the agent skill sets and routing rules are given. The routing policies we consider are of the (static) *overflow routing* family: Each call class has an ordered list of agent types that can handle it; upon arrival, a call of that class is assigned to the first agent type in this list that has an available agent, or else is placed in queue (one queue per call class). Likewise, each agent type has an ordered list of call classes (queues) from which to pick up calls when it becomes available. The problem is to minimize staffing costs subject to a set of constraints on service levels (globally and per call class), assuming the center is in steady-state. This problem was proposed to us by Bell Canada, a Canadian company whose call centers serving Quebec and Ontario employ nearly 13,000 agents in total. The service level constraints are quite important to them because governmental regulations impose huge fines to the company when some of these constraints are not met on average over the month. The static routing rules may be seen as restrictive, but they were also a request from the company; they wanted to have a tool telling

them what happens when they optimize under such constraints, with fixed routings and skill sets. In general, lower costs can obviously be achieved by relaxing the routing rules and optimizing the skill sets, so we do not necessarily recommend fixing these in practice. On the other hand, it is not always possible to have agents with *any* (arbitrary) combination of skills. Single-period staffing appears as a subproblem in several scheduling algorithms (Gans et al., 2003; Bhulai et al., 2008).

After formulating this problem as an integer program with linear objective function and non-linear constraints, Cezik and L'Ecuyer (2008) developed and studied a general-purpose solution approach, based on ideas adapted from Atlason et al. (2004). They use integer programming with cutting planes; the cuts are obtained by estimating subgradients of the service-level constraints with respect to the decision variables (the number of agents of each type) via simulation. This method can handle arbitrarily complex call-center operating conditions (e.g., call-routing policy, nonstationarity, etc.). On the other hand, subgradient estimation by simulation is very time-consuming. Accepting noisier estimates obtained from shorter simulations can save time, but is more likely to return highly suboptimal or infeasible solutions (Cezik and L'Ecuyer, 2008).

Our approach aims to be a quicker alternative by exploiting approximations of the service level. However, in the multi-skill setting, good approximations are not generally available. If we assume that the call center is a *loss system*, i.e., there are no waiting queues and all calls that cannot be served immediately are lost, then there are many approximations of the loss (or *blocking*) probability per call class. Koole and Talim (2000) assume overflow routing and develop an approximation via decomposition into subsystems whose state space is smaller and which are easier to analyze. Koole et al. (2003) allow queueing and approximate the delay probability (i.e., that delay is positive), based on this loss approximation and the relation between the loss probability in the Erlang B system and the delay probability of the Erlang C system. This relation involves the staffing, so their formula applies globally, but not per class. Better estimates of loss probabilities can be obtained via two-moment approximations of the overflow process (the *equivalent random method*, or Hayward's approximation) (Cooper, 1981; Wolff, 1989; Chevalier et al., 2003, 2004); and the method of Franx et al. (2006). These better methods restrict the overflow pattern (it cannot be *cyclical*, as defined in Section 3.1). One could choose to approximate the (class-specific) service

level in a real system (i.e., with queueing) by the approximated loss probability in a relevant loss model. This would make sense in a system where most calls do not wait. However, this approach is unnatural for most modern call centers, which normally operate so that the fraction of calls that experience a positive delay in queue is considerable (Gans et al., 2003).

Our first contribution is an approximation of the service level per class in a multi-skill center with a special type of overflow routing. This *loss-delay* (LD) approximation exploits ideas from Koole and Talim (2000) and goes beyond a loss system by incorporating queueing. Essentially, it assumes that whenever a call is delayed, it waits in a queue for the last agent type (skill set) in its list. The approximation has accuracy that varies across problems; despite this, we show that it is useful as a support tool in a staffing-cost minimization algorithm. We do this via examples where the routing policy has the overflow element but does not satisfy the latter assumption (waiting at the last agent type). The routing policy was specified by our industrial partner.

Our second contribution is a heuristic approach to the staffing problem. Key components are appropriate initialization and neighborhood search methods supported by the LD approximation in deciding neighbor feasibility and in selecting to which feasible neighbor to move. The first stage of the search terminates with a solution that is locally optimal (relative to a certain neighborhood) after a finite amount of work. The second stage uses local-search procedures supported by estimates of service levels that are more accurate than the LD approximation and are obtained by simulation. These procedures adjust the solution for feasibility or further cost reduction, evaluating only few additional solutions. We solve realistic problems of varying size and find that our approach often yields better solutions than that of Cezik and L'Ecuyer (2008) when the computing budget is limited. Although none of the two methods always dominates the other, the new heuristic is definitely a useful addition to the toolbox for this class of problems.

We mention other related work. Harrison and Zeevi (2005) and Bassamboo et al. (2006) consider a call center with a doubly stochastic time-varying arrival processes in an asymptotic regime and find a staffing and routing policy that asymptotically minimizes the cost of staffing and abandonment. It is unclear how their fluid approximation could provide good estimates of the service levels (to solve our problem). Wallace and Whitt (2005) assume no constraints on the skill sets, except that

each agent has exactly two skills. They optimize *both* the staffing and the skill sets, simultaneously. They allow a different (and more flexible) routing rule than ours and assume that all agent types cost the same. For staffing under a single service-level constraint, Pot et al. (2008) have a heuristic that uses a line search to optimize the Lagrange multiplier for the constraint. It is unclear how this can be generalized to an efficient algorithm when there are multiple constraints.

The remainder is organized as follows. In Section 2 we formulate mathematical programs of multi-skill staffing and scheduling and review related literature. Section 3.1 defines overflow routing and a related policy. Sections 3.2 to 3.4 develop the LD approximation. Section 4 describes our approach to the staffing problem and Section 5 details the solution of several problem instances. In Section 6 we compare to alternative approaches, including adapting the method of Wallace and Whitt (2005) and replacing our approximation by that of Koole and Talim (2000). Additional details of our approach are contained in the Online Appendix; this includes detailed algorithms and an assessment of sensitivity to algorithm parameters.

2 Formulation of staffing and scheduling problems

The sets of call classes and agent types are $\mathcal{N} = \{1, \ldots, n\}$ and $\mathcal{M} = \{1, \ldots, m\}$, respectively. There are b time periods and s types of *shifts*; a shift is defined by specifying the time periods in which the agent is available to handle calls. The *cost vector* is $\mathbf{c} = (c_{1,1}, \ldots, c_{1,s}, \ldots, c_{m,1}, \ldots, c_{m,s})^{\mathrm{T}}$, where $c_{i,q}$ is the cost of an agent of type i having shift q, and "T" denotes vector transposition. Write $z_{i,q}$ for the number of agents of type i having shift q and set $\mathbf{z} = (z_{1,1}, \ldots, z_{1,s}, \ldots, z_{m,1}, \ldots, z_{m,s})^{\mathrm{T}}$. Write $x_{i,p}$ for the number of agents of type i that are available to handle calls in period p. Then the *staffing vector* $\mathbf{x} = (x_{1,1}, \ldots, x_{1,b}, \ldots, x_{m,1}, \ldots, x_{m,b})^{\mathrm{T}}$ satisfies $\mathbf{x} = \mathbf{A}\mathbf{z}$ where $\mathbf{A}$ is a block-diagonal matrix with m identical blocks $\tilde{\mathbf{A}}$, where the element (p, q) of $\tilde{\mathbf{A}}$ is 1 if shift q covers period p, and 0 otherwise. Our definition of the *service level* (SL) of call class j during period p is

$$g_{j,p}(\mathbf{x}) = \frac{\mathbb{E}[\#\text{ of type-}j\text{ call arrivals in }p\text{ that are served and wait at most }\tau]}{\mathbb{E}[\#\text{ of type-}j\text{ call arrivals in }p\text{, except those that wait less than }\tau\text{ and abandon}]} \quad (1)$$

where "abandon" implies the call joined the queue—it was not lost immediately upon arrival; and τ is a constant called the *acceptable waiting time* (AWT). In Online Appendix A.1, we consider an

alternative SL definition in which calls that are delayed by less than τ and abandon are not excluded; and we provide formulas for its approximation. In our examples, the two measures of service level differed by at most 2% in moderate-abandonment cases and negligibly in low-abandonment cases. Given acceptable waiting times τ_p, τ_j, and τ, aggregate SLs are defined analogously and denoted $g_p(\mathbf{x})$, $g_j(\mathbf{x})$ and $g(\mathbf{x})$ for period p, call class j, and overall, respectively.

A formulation of the *scheduling problem* is

$$\begin{array}{rll} \min & \mathbf{c}^{\mathrm{T}}\mathbf{z} = \sum_{i=1}^{m}\sum_{q=1}^{s} c_{i,q}z_{i,q} & \\ \text{subject to} & \mathbf{A}\mathbf{z} = \mathbf{x}, & \\ & g_{j,p}(\mathbf{x}) \geq l_{j,p} & \text{for all } j,p, \\ & g_p(\mathbf{x}) \geq l_p & \text{for all } p, \\ & g_j(\mathbf{x}) \geq l_j & \text{for all } j, \\ & g(\mathbf{x}) \geq l, & \\ & \mathbf{z} \geq 0, \text{ and integer.} & \end{array} \tag{P1}$$

where $l_{j,p}$, l_p, l_j and l are given constants. The *staffing problem* is a *relaxation* of the scheduling problem where we assume that any staffing $\mathbf{x}$ is admissible. In a single-period staffing problem, we have $b = 1$, $\mathbf{c} = (c_1, \ldots, c_m)^{\mathrm{T}}$, where c_i is the cost of an agent of type i, and $\mathbf{x} = (x_1, \ldots, x_m)^{\mathrm{T}}$, where x_i is the number of agents of type i. The optimization problem then reduces to:

$$\begin{array}{rll} \min & \mathbf{c}^{\mathrm{T}}\mathbf{x} = \sum_{i=1}^{m} c_i x_i & \\ \text{subject to} & g_j(\mathbf{x}) \geq l_j & \text{for all } j, \\ & g(\mathbf{x}) \geq l, & \\ & \mathbf{x} \geq 0, \text{ and integer.} & \end{array} \tag{P2}$$

In the presence of abandonments, the SL functions $g_\bullet$ are typically S-shaped in each coordinate, i.e., convex increasing below a certain threshold, and concave increasing above the threshold (Henderson and Mason, 1998; Ceẓik and L'Ecuyer, 2008). Adapting the method of Atlason et al. (2004), Ceẓik and L'Ecuyer (2008) approximate optimal solutions of (P2) by (exact or approximate) solutions to some analog of (P2), called the *sample problem*, defined by replacing each $g_\bullet$ by a noisy estimate that is computed by simulation and is called the *sample service level* function. Their algorithm involves

iterative solution of integer programs and addition of *cuts* (linear inequalities), each one being derived from an estimate of a subgradient of some $g_\bullet$, where the subgradient estimate is computed via finite differences of the corresponding sample service level. The approach is heuristic: the cuts sometimes eliminate subsets of the feasible set that include the optimal solution, because of the noise in the estimates and also because these subgradient estimates would not necessarily be true subgradients even if the simulation-estimation error were to vanish. Cezik and L'Ecuyer (2008) suggest practical heuristics around this and other problems.

This paper address the solution of (P2) only. Solving (P2) is a possible first step in solving (P1). This is the approach taken in Pot et al. (2008).

3 Performance approximation under overflow routing

In this section we develop the loss-delay approximation of the service levels. We analyze an overflow-type policy in which whenever a call is delayed, it waits in a queue served only by the last agent type on the list. Under this policy, the approximation arises naturally. We do not claim that this policy is efficient or that it is found in practice. We emphasize that the models we optimize do not need to have the wait-at-the-last-agent-type feature.

3.1 Overflow routing and approximation overview

We refer to *station* i as the ensemble of type-i agents. Agents within a station are indistinguishable. For each call class j we have a list (an ordered set) of stations. *Overflow routing* means that upon arrival, a class-j call is assigned to the first station in the list that has an available agent or else is placed in queue. Whenever the assigned station is not the first one on the list, we say that an *overflow* has occurred from station ranking $l-1$ on the list to station ranking l on the list, for each relevant l. The *overflow-or-wait-at-last-station* policy specifies additionally that each delayed call is served only at the last station in its list.

As background for computational issues in Section 3.4, we characterize the routing by a directed *flow graph*. The flow graph has a vertex for each station. All possible overflows from one station to another are represented by directed arcs. A routing is called *crossed* whenever the flow graph has

a directed cycle. Such a situation might arise as follows. Call class 1 has high revenue-generation potential; call class 2 has a service nature and low revenue-generation potential. Type-A agents are stronger in selling services, and type-B agents are stronger in servicing. A goal of maximizing the flow of class-1 calls to type-A agents would motivate the list $\{A, B\}$ for class 1 and the reverse list for class 2. Thus, the flow graph has a directed cycle between vertices A and B.

Here is an outline of the approximation. For each station i, the set of call classes that can be served there is partitioned into two sets: $\mathcal{L}_i$ contains classes that can overflow into another station; and $\mathcal{D}_i$ contains classes for which no overflow is possible (i.e., i is the last station on the call's list). Whenever both these sets are non-empty, i is a *loss-delay* station. Otherwise, i is a *loss* station when all classes can overflow and a *delay* station when no class can overflow. Our basic building block is the analysis of a loss-delay station. The sets $\mathcal{L}_i$ and $\mathcal{D}_i$ define respective arrival streams; when no server is available, calls in the first stream can overflow, but the ones in the second must queue for service in this station. We approximate these streams as independent Poisson processes and analyze the station as a one-dimensional birth and death process. This is detailed in Sections 3.2 and 3.3, where we allow and exclude customer abandonment, respectively. We obtain two related approximations: LDA (abandonment) and LDN (no abandonment). The absence of abandonment makes the second one less realistic but considerably faster to compute.

3.2 Analysis of a loss-delay station with abandonment

The station has s servers and a queue with capacity c. Calls of types *delay* and *loss* arrive according to independent Poisson processes with rates λ_{D} and λ_{L}, respectively. Service times of delay and loss calls are i.i.d. exponential random variables with mean $1/\mu_{\mathrm{D}}$ and $1/\mu_{\mathrm{L}}$, respectively, and independent of everything else. Server preemptions are not allowed. Loss calls that cannot be immediately served upon arrival are lost. Delay calls abandon as soon as their time in queue equals their *patience time*. Patience times are i.i.d exponential random variables with mean $1/\eta$, independent of everything else. Delay calls that find c calls in queue upon arrival are lost.

Consider first the case $\mu_{\mathrm{L}} = \mu_{\mathrm{D}} = \mu$. Let $X(t)$ denote the number of calls in the station at time t; this is the sum of loss calls in service and delay calls in the system, i.e., either in service

or in queue. The process $X = \{X(t) : t \geq 0\}$ is a birth and death process with finite state space, $\{0, 1, 2, \ldots, s+c\}$; the birth rates λ_k and death rates μ_k are:

$$\begin{aligned} \lambda_k &= \begin{cases} \lambda_{\mathrm{D}} + \lambda_{\mathrm{L}}, & k = 0, 1, \ldots, s-1 \\ \lambda_{\mathrm{D}}, & k = s, s+1, \ldots, s+c-1 \end{cases} \\ \mu_k &= \begin{cases} k\mu, & k = 1, 2, .., s \\ s\mu + (k-s)\eta, & k = s+1, s+2 \ldots, s+c. \end{cases} \end{aligned}$$

The stationary probabilities, $\pi_k = \lim_{t\to\infty} \Pr\{X(t) = k\}$, are $\pi_k(\mu) = \pi_0 \prod_{i=1}^{k} (\lambda_{i-1}/\mu_i)$, $k = 1, 2, \ldots, s+c$, where $\pi_0(\mu) = \left(1 + \sum_{k=1}^{s+c} \prod_{i=1}^{k} \lambda_{i-1}/\mu_i\right)^{-1}$ (Ross, 1983, p. 154).

Our approximation for the general case ($\mu_{\mathrm{L}} \neq \mu_{\mathrm{D}}$) is based on an "effective" service rate found by equating the input average service time to the output average service time determined by the mix of service completions of the two types. To this end, note that delay-call service completions occur at the rate $\tilde{\lambda}_{\mathrm{D}}(\mu) = \lambda_{\mathrm{D}}[1 - \pi_{s+c}(\mu)] - \eta \sum_{k=1}^{c-1} k\pi_{s+k}(\mu)$, by counting arrivals minus losses due to a full queue, minus losses via abandonment, and using PASTA (Poisson Arrivals See Time Averages) (Wolff, 1989). Thus, the effective service rate μ^* must be a root of the function

$$h_1(\mu) = \frac{w_1(\mu)}{\mu_{\mathrm{D}}} + \frac{1 - w_1(\mu)}{\mu_{\mathrm{L}}} - \frac{1}{\mu}, \tag{2}$$

where $w_1(\mu) = \tilde{\lambda}_{\mathrm{D}}(\mu)/[\tilde{\lambda}_{\mathrm{D}}(\mu) + \lambda_{\mathrm{L}}(1 - B_{\mathrm{A}}(\mu))]$ is the stationary fraction of service completions that are of delay type and $B_{\mathrm{A}}(\mu) = \sum_{k=s}^{s+c} \pi_k(\mu)$ is the blocking probability. The existence of a root and a simple algorithmic solution follow from:

Proposition 1 *The function h_1 has at least one root in $\mathcal{J} = [\min(\mu_{\mathrm{L}}, \mu_{\mathrm{D}}), \max(\mu_{\mathrm{L}}, \mu_{\mathrm{D}})]$.*

Proof. It is easy to check that the function h_1 is continuous with $h_1(\min(\mu_{\mathrm{L}}, \mu_{\mathrm{D}})) < 0$ and $h_1(\max(\mu_{\mathrm{L}}, \mu_{\mathrm{D}})) > 0$. □

We use a result on the *virtual waiting time*, defined as the waiting time in queue that would be spent by an infinitely patient customer. Write W for the stationary virtual waiting time.

Lemma 1 (Riordan, 1962, p. 110-111) *Given that the system state upon arrival is X, we have*

$$p_k(\mu, \tau) = \Pr\{W > \tau \,|X = s + k\} = \xi^{\phi} \sum_{j=0}^{k} \frac{(\phi)_j (1-\xi)^j}{j!}, \quad \tau > 0, k \geq 0, \tag{3}$$

where $\phi = s\mu/\eta$, $(\phi)_0 = 1$, $(\phi)_j = (\phi)(\phi+1)\cdots(\phi+j-1)$ for $j \geq 1$, and $\xi = e^{-\eta\tau}$.

The probability that a delay call is lost upon arrival or its virtual waiting time exceeds τ is

$$D_{\mathrm{A}}(\tau) = \pi_{s+c}(\mu^*) + \sum_{k=0}^{c-1} \pi_{s+k}(\mu^*) p_k(\mu^*, \tau). \tag{4}$$

Later, we combine this measure of service (at the last station) with the probability of overflow to this station (when applicable). This measure is not entirely consistent with the SL in (1). In Online Appendix A.1 we provide an approximation that is conceptually consistent with (1). In our examples, the difference between the two approximation values was small and did not appear to affect any of our conclusions. We prefer the one presented here because it is much faster to compute. The long-run fraction of delay calls that abandon is $1 - \tilde{\lambda}_{\mathrm{D}}(\mu^*)/\lambda_{\mathrm{D}}$. Pure-loss and pure-delay stations are the special cases $\lambda_{\mathrm{D}} = 0$ and $\lambda_{\mathrm{L}} = 0$, respectively.

3.3 Analysis of a loss-delay station without abandonment

We modify the setup of section 3.2, now specifying no abandonment and an infinite queue capacity. The analysis is similar, so we only state the main formulas. If $\mu_{\mathrm{L}} = \mu_{\mathrm{D}} = \mu$ and $\lambda_{\mathrm{D}} < s\mu$, then X is a birth and death process with infinite state space, $\{0,1,2,...\}$. Here, the work needed to compute the stationary distribution is $O(s)$ because the stationary probabilities of states above s decay geometrically. This contrasts with the model with abandonment, where there is no geometric structure, the same task takes $O(s+c)$ work, and a finite queue capacity is a necessity. The blocking probability is $B(\mu) = \pi_s s\mu/(s\mu - \lambda_{\mathrm{D}})$, where $\pi_s = \pi_0 \rho^s/s!$ with $\rho = (\lambda_{\mathrm{L}} + \lambda_{\mathrm{D}})/\mu$ and $\pi_0 = \left\{\sum_{k=0}^{s-1} \rho^k/k! + (\rho^s/s!)s\mu/(s\mu - \lambda_{\mathrm{D}})\right\}^{-1}$.

For the general case $\mu_{\mathrm{D}} \neq \mu_{\mathrm{L}}$, we again find an effective service rate. If $\lambda_{\mathrm{D}} \geq s\mu_{\mathrm{D}}$, then X is not positive recurrent and the station is *unstable*. In the remainder, assume $\lambda_{\mathrm{D}} < s\mu_{\mathrm{D}}$. Define $h(\mu) = w(\mu)/\mu_{\mathrm{D}} + (1 - w(\mu))/\mu_{\mathrm{L}} - 1/\mu$ for $\mu > \lambda_{\mathrm{D}}/s$, where $w(\mu) = \lambda_{\mathrm{D}}/[\lambda_{\mathrm{D}} + \lambda_{\mathrm{L}}(1 - B(\mu))]$, and define $\mathcal{I} = (\mu_1, \mu_2]$, with $\mu_1 = \max\left(\lambda_{\mathrm{D}}/s, \min(\mu_{\mathrm{L}}, \mu_{\mathrm{D}})\right)$ and $\mu_2 = \max(\mu_{\mathrm{L}}, \mu_{\mathrm{D}})$. We have:

Proposition 2 *Assume $\lambda_{\mathrm{D}} < s\mu_{\mathrm{D}}$. For any μ in $\mathcal{I}$, the process X is positive recurrent. The function h has at least one root in $\mathcal{I}$. If $\mu_{\mathrm{D}} > \mu_{\mathrm{L}}$, then the root is unique.*

The proof of Proposition 2 is in Online Appendix A.4. The counterparts of functions B_{A} and D_{A} (of Section 3.2) are $B(\mu^*)$ and $D(\tau; s, \lambda_{\mathrm{L}}, \lambda_{\mathrm{D}}, \mu_{\mathrm{L}}, \mu_{\mathrm{D}}) = B(\mu^*)e^{-\tau(s\mu^* - \lambda_{\mathrm{D}})}$, respectively, where μ^* is

a root of h. The function D above shows all the inputs and is referred to in Online Appendix A.1.

3.4 The loss-delay approximation

For each call class j, we have arrival rate λ_j, abandonment rate η_j, and the routing list (ordered set) $\mathcal{R}_j$. The service rate of type j at station i is $\mu_{i,j}$. We also have a staffing $\mathbf{x} = (x_i)_{i=1}^m$. The term *arrival* to a station encompasses both exogenous arrivals and overflows. We approximate: the process of type-j arrivals to station i as being Poisson with rate $\gamma_{i,j}$; and the blocking probability, B_i, whenever i is a pure-loss or loss-delay station. Write $p(i,j)$ for the station immediately preceding i in the list of call class j. The LDA approximation requires:

$$\gamma_{i,j} = \begin{cases} \lambda_j, & \text{whenever } i \text{ is first in } \mathcal{R}_j \\ \gamma_{p(i,j),j} B_{p(i,j)}, & \text{whenever } p(i,j) \text{ exists} \end{cases} \tag{5}$$

$$\gamma_{i,\mathrm{L}} = \sum_{j \in \mathcal{L}_i} \gamma_{i,j}, \quad \frac{1}{\mu_{i,\mathrm{L}}} = \sum_{j \in \mathcal{L}_i} \frac{\gamma_{i,j}}{\gamma_{i,\mathrm{L}}} \frac{1}{\mu_{i,j}} \qquad \text{whenever } \mathcal{L}_i \text{ is non-empty,} \tag{6}$$

$$\gamma_{i,\mathrm{D}} = \sum_{j \in \mathcal{D}_i} \gamma_{i,j}, \quad \frac{1}{\mu_{i,\mathrm{D}}} = \sum_{j \in \mathcal{D}_i} \frac{\gamma_{i,j}}{\gamma_{i,\mathrm{D}}} \frac{1}{\mu_{i,j}} \quad , \tilde{\eta}_i = \sum_{j \in \mathcal{D}_i} \frac{\gamma_{i,j}}{\gamma_{i,\mathrm{D}}} \eta_j \quad \text{whenever } \mathcal{D}_i \text{ is non-empty} \tag{7}$$

$$B_i = B_{\mathrm{A}}\left(x_i, \gamma_{i,\mathrm{L}}, \gamma_{i,\mathrm{D}}, \mu_{i,\mathrm{L}}, \mu_{i,\mathrm{D}}, \tilde{\eta}_i, c_i\right) \quad \text{whenever } \mathcal{L}_i \text{ is non-empty,} \tag{8}$$

where: B_{A} is the blocking probability from Section 3.2 (the notation here shows all the function inputs); and $c_i = \max(\lceil \psi \sqrt{x_i} \rceil, 10)$, where ψ is a queue-size control parameter (this formula is motivated later). The *overflow equations* in (5) state that the class-j overflow rate to station i equals the class-j arrival rate to the station immediately preceding i in the routing times the blocking probability at that station. In (7), parameters $\gamma_{i,\mathrm{D}}$, $\mu_{i,\mathrm{D}}$, and $\tilde{\eta}_i$ of the aggregate delay stream are based on the analogous parameters of the constituent call classes; and similarly in (6) for the loss stream. The LDA approximation of class-j SL, with AWT τ_j, is

$$\hat{g}_j(\mathbf{x}, \tau_j) = 1 - \frac{\gamma_{\ell(j),j}}{\lambda_j} D_{\ell(j)}(\tau_j), \quad \tau_j > 0, \tag{9}$$

where $\ell(j)$ is the last station in $\mathcal{R}_j$ and $D_{\ell(j)}(\cdot)$ is the function (4) applied to this station. The global service level approximation is $\hat{g}(\mathbf{x}, \tau) = \sum_{j \in \mathcal{N}} \lambda_j \hat{g}_j(\mathbf{x}; \tau) / (\sum_{j \in \mathcal{N}} \lambda_j)$. LDA approximations of other common performance measures are given in Online Appendix A.0. The approximated arrival (overflow) rate of each call type to its last station parallels that of Koole and Talim (2000)

(KT); the overflow equations (5) are the same, except that the blocking probabilities differ in upstream stations having a delay stream.

The LDN approximation makes obvious modifications: we replace the functions $B_{\rm A}$ and $D_{\rm A}$ by their counterparts B and D in Section 3.3, respectively.

Several steps are heuristic. The two aggregate arrival streams at each station are treated as being Poisson and independent, which typically fails to hold when overflows are involved. Taking weighted averages of service-time means and patience-time rates in (6) and (7) is a heuristic (to keep the birth and death state space one-dimensional). To motivate our choice to average means in one case and rates in the other case, suppose first that we have two classes with abandonment rates η_j, $j = 1, 2$ and weight $1/2$ each, where $\eta_1 = 1$. Let ϵ be a number small compared to 1, and consider first the case $\eta_2 = \epsilon$. The average patience is $(1/\epsilon + 1)/2 \approx 1/(2\epsilon)$, implying the small rate 2ϵ, whereas the average rate is $(\epsilon + 1)/2 \approx 1/2$; as $\epsilon \to 0$, averaging rates seems preferable. The situation is reversed if $\eta_2 = 1/\epsilon$ (i.e., large compared to 1): the average patience, $(\epsilon + 1)/2 \approx 1/2$, implies the rate 2, whereas the average rate, $(1/\epsilon + 1)/2 \approx 1/(2\epsilon)$, is large; here, as $\epsilon \to 0$, averaging means seems preferable. However, this second situation is expected to be less common in call centers, since the fraction of calls who abandon is usually very small. Also note that $(\sum_i w_i \mu_i^{-1})^{-1} < \sum_i w_i \mu_i$ whenever $0 < w_i < \infty$, $0 < \mu_i < \infty$, and the μ_i's are not all equal (by Jensen's inequality). Thus, averaging service-time means is conservative relative to averaging rates. The formula $c_i = \max(\psi\sqrt{x_i}, 10)$ following (8) aims for economy of computation relative to setting a large c_i for all stations. A rough justification is that the stationary number of customers in an infinite-server Markovian queue is Poisson-distributed (so the standard deviation equals the square root of the mean) and the anticipation that in solutions of interest, each x_i is of the order of this mean. The idea, then, is that we hope to make c_i a state of small probability for each i whenever x_i is moderately large (say, 25 or more) by appropriate selection of ψ (in our examples, we set $\psi = 2$). For the x_i that fail this rough criterion, the infinite-server argument is less reliable, so we imposed the lower bound 10 at the outset.

Figure 1 specifies an iterative algorithm that converges, under certain conditions, to a solution to (5)–(8). In summary, the algorithm initializes the overflow rates to zero and computes iteratively

(5)–(8) until the change in the blocking probabilities is deemed small enough. Koole and Talim (2000) employ a similar technique in a two-station loss system. In the LDN approximation, if for some k and i we have $\gamma_{i,\mathrm{D}}^{(k)} \geq x_i \mu_{i,\mathrm{D}}^{(k)}$, then station i and the system are declared *indeterminate*; if the flow graph is acyclic, this means that station i is unstable in the model of Section 3.3. When running this algorithm, we may have to settle for an approximate solution by relaxing the convergence criterion. We do this as follows: if (10) fails at $k = k_{\mathrm{U}}$, then we double k_{U} and reiterate (go to step 2); if (10) fails again at the doubled k_{U}, then we reiterate for at most 10 iterations, doubling ϵ in each iteration. In our experiments, this happened only on rare occasions.

In Proposition 3, we prove that the algorithm of Figure 1 converges to a solution of (5)–(8), under certain conditions on the service rates and abandonment rates. These conditions imply that the rates $\gamma_{i,j}^{(k)}$ and the blocking probability $B_i^{(k)}$ are monotone increasing in k, and the convergence proof then follows by exploiting this monotonicity in an induction argument. Without these conditions (and the monotonicity), a convergence proof appears more complicated, but we think that convergence to a solution should occur in most practical cases (where there are abandonments).

Proposition 3 *Suppose that for each station, the classes in the loss and delay streams have common service rates.*

1. *(LDA approximation.) Suppose that for each station, the classes in the delay stream have common abandonment rate. Then there exist limits $\gamma_{i,j} = \lim_{k\to\infty} \gamma_{i,j}^{(k)}$, and likewise for all other quantities with superscripts in Figure 1. The limits satisfy (5)–(8).*

2. *(LDN approximation.) Replace the function B_{A} in Figure 1 by its counterpart B. Unless the system is declared indeterminate, the analogous limits exist and solve the analogous equations.*

Proof. For part 2, we assume the system is not declared indeterminate (otherwise, there is nothing to prove). Define $\Delta\gamma_{i,j}^{(k)} = \gamma_{i,j}^{(k)} - \gamma_{i,j}^{(k-1)}$ and $\Delta B_\ell^{(k)} = B_\ell^{(k)} - B_\ell^{(k-1)}$.

First, we prove by induction on k that

$$\Delta\gamma_{i,j}^{(k)} \geq 0 \text{ and } \Delta B_\ell^{(k)} \geq 0 \quad \text{for all } i, j, \ell \text{ and for all } k \geq 1. \tag{11}$$

Algorithm LD:

Restriction: $\mu_j > 0$ and $\eta_j > 0$ for all j.
Select: ϵ (convergence tolerance), k_{U} (number of iterations), ψ (controls queue size).

1. $\gamma_{i,j}^{(0)} = 0$ whenever $p(i,j)$ exists
   $\gamma_{i,j}^{(k)} = \lambda_j$ for all k, whenever i is first in $\mathcal{R}_j$
   $B_i^{(0)} = 0$ for all i
   $k = 1$.

2. Compute iterated overflows:

$$\gamma_{i,j}^{(k)} = \gamma_{p(i,j),j}^{(k-1)} B_{p(i,j)}^{(k-1)} \quad \text{whenever } p(i,j) \text{ exists.}$$

3. Aggregate parameters by station and stream type (loss or delay):

$$\gamma_{i,\mathrm{L}}^{(k)} = \sum_{j \in \mathcal{L}_i} \gamma_{i,j}^{(k)}, \quad \frac{1}{\mu_{i,\mathrm{L}}^{(k)}} = \sum_{j \in \mathcal{L}_i} \frac{\gamma_{i,j}^{(k)}}{\gamma_{i,\mathrm{L}}^{(k)}} \frac{1}{\mu_{i,j}} \qquad \text{whenever } \mathcal{L}_i \text{ is non-empty.}$$

$$\gamma_{i,\mathrm{D}}^{(k)} = \sum_{j \in \mathcal{D}_i} \gamma_{i,j}^{(k)}, \quad \frac{1}{\mu_{i,\mathrm{D}}^{(k)}} = \sum_{j \in \mathcal{D}_i} \frac{\gamma_{i,j}^{(k)}}{\gamma_{i,\mathrm{D}}^{(k)}} \frac{1}{\mu_{i,j}}, \quad \tilde{\eta}_i^{(k)} = \sum_{j \in \mathcal{D}_i} \frac{\gamma_{i,j}^{(k)}}{\gamma_{i,\mathrm{D}}^{(k)}} \eta_j \quad \text{whenever } \mathcal{D}_i \text{ is non-empty.}$$

4. Compute blocking probabilities:

$$B_i^{(k)} = B_{\mathrm{A}}\left(x_i, \gamma_{i,\mathrm{L}}^{(k)}, \gamma_{i,\mathrm{D}}^{(k)}, \mu_{i,\mathrm{L}}^{(k)}, \mu_{i,\mathrm{D}}^{(k)}, \tilde{\eta}_i^{(k)}, c_i\right) \quad \text{whenever } \mathcal{L}_i \text{ is non-empty.}$$

   where $c_i = \max(\lceil \psi \sqrt{x_i} \rceil, 10)$.

5. Convergence criterion:

$$\max_{i:\mathcal{L}_i \neq \emptyset} \left| B_i^{(k)} - B_i^{(k-1)} \right| < \epsilon. \tag{10}$$

   If (10) holds, then return the current rates $\gamma_{i,j}^{(k)}$ as an approximate solution;
   else, if $k < k_{\mathrm{U}}$, then $k \leftarrow k+1$ and go to 2;
   else, $k \leftarrow k+1$, increase k_{U} and/or increase ϵ and go to 2.

Figure 1: Algorithm to compute an exact or approximate solution for staffing $(x_i)_{i=1}^m$.

This is obviously true for $k = 1$. Assume that (11) holds for a given k. Observe that whenever i is first in $\mathcal{R}_j$, we have $\Delta\gamma_{i,j}^{(k)} = 0$ for all k. Otherwise (i.e., if $p(i,j)$ exists), for all $k \geq 1$, we have

$$\Delta\gamma_{i,j}^{(k+1)} \;=\; \gamma_{p(i,j),j}^{(k)} B_{p(i,j)}^{(k)} - \gamma_{p(i,j),j}^{(k-1)} B_{p(i,j)}^{(k-1)} \;=\; \gamma_{p(i,j),j}^{(k)} \Delta B_{p(i,j)}^{(k)} + B_{p(i,j)}^{(k-1)} \Delta\gamma_{p(i,j),j}^{(k)} \;\geq\; 0.$$

The non-negativity of $\Delta\gamma_{i,j}^{(k+1)}$ implies $\gamma_{i,\mathrm{L}}^{(k+1)} - \gamma_{i,\mathrm{L}}^{(k)} \geq 0$ and $\gamma_{i,\mathrm{D}}^{(k+1)} - \gamma_{i,\mathrm{D}}^{(k)} \geq 0$ for all i. Combining this with our assumptions on common service rates and common abandonment rates, and the fact that the functions B_{A} and B are increasing in λ_{L} and λ_{D} when all other arguments are fixed, we obtain that $\Delta B_i^{(k+1)} \geq 0$ for all i, in LDA and LDN. This completes the induction.

Since the sequence $\{\gamma_{i,j}^{(k)}\}_{k=1}^{\infty}$ is obviously upper-bounded by λ_j (easily seen by induction on k), $\lim_{k\to\infty} \gamma_{i,j}^{(k)}$ must exist for each i and j, in LDA and LDN. Moreover, since each $B_i^{(k)}$ is a continuous function of $\gamma_{i,\mathrm{L}}^{(k)}$, $\gamma_{i,\mathrm{D}}^{(k)}$, $\mu_{i,\mathrm{L}}^{(k)}$, $\mu_{i,\mathrm{D}}^{(k)}$, and $\tilde{\eta}_i^{(k)}$, and since these are continuous functions of certain $\gamma_{i,j}^{(k)}$, all these sequences have limits that together satisfy (5)–(8). $\square$

Algorithm LD can generally be made more efficient. In the special case of an acyclic flow graph, it can even be streamlined to give the exact solution in a single iteration. This is only summarized here; see Chan (2006) for a complete treatment. In the general case, one can first partition the flow graph into its *strongly connected components*, via standard algorithms (Aho et al., 1974, p. 189-195). Second, one finds a permutation Π of the components such that all overflows occur along increasing Π value. If the flow graph is acyclic (each component is one of the stations), then the solution is unique and can be computed by executing (5)–(8) in the order $i = \Pi(1)$, $\Pi(2), \ldots, i = \Pi(m)$. Otherwise, it suffices to apply a restricted version of Algorithm LD to each of the components, ordered along increasing Π value. The assumptions of Proposition 3 can be weakened: to ensure convergence to a solution, it suffices to have common parameters in each station within a component, but not necessarily across components.

4 Multi-skill staffing by search methods

Our method is supported by an evaluator of service levels. An *incumbent*, i.e., current solution, is maintained throughout. A solution is called *E-(in)feasible* and *SIM-(in)feasible* depending on its (in)feasibility for (P2), as deemed by the evaluator and simulation, respectively; in general, these

Algorithm Staff:

Stage 0. (Initialization) Construct a solution that is E-feasible.

Stage 1. (Neighborhood search). Until a termination condition is true, do:

1a. Select a positive integer *move size* q.

1b. Consider removing q agents of the same type, for each possible type; if at least one of these solutions is E-feasible, then select a new incumbent among them and repeat the Until statement.

1c. Consider removing q agents of some type and adding q agents of a less expensive type; if at least one of these solutions is E-feasible, then select a new incumbent among them.

Stage 2a. If necessary, adjust the incumbent toward SIM-feasibility.

Stage 2b. Attempt to reduce the cost of the SIM-feasible incumbent.

Figure 2: Outline of the staffing algorithm

do not coincide with exact (in)feasibility. An outline of the staffing algorithm appears in Figure 2.

We now discuss the algorithm components, leaving out details to pseudocodes in Online Appendix A.1. Solution vectors are denoted $\mathbf{x}$, where x_i is the i-th component; $\mathbf{e}_j$ is the j-th unit m-vector.

Stage 0: Initialization. Our method is as follows: (1) For each call class, allocate the arrival rate to the feasible stations: send a fraction β ($0 \leq \beta \leq 1$) to the cheapest one and split the remaining fraction evenly among the others; (2) Compute parameters of the aggregate arrival stream in each station, based on step 1; (3) Viewing each station as Markovian ($M/M/s/M$) and independent of all others, set the staffing to the minimal one that achieves an SL of at least ξ; and (4) If necessary, iteratively increase this solution to obtain an E-feasible one. The rationale is to roughly control the total number of agents via ξ and the fraction of low-cost agents via β. In all our experiments, setting $\xi = l$ (the global SL target) yielded an E-feasible solution after step 3, so step 4 was unnecessary. The details, including two alternatives for step 4, are in Procedure `Init` in Online Appendix A.1. Online Appendix A.2 contains experimental results for this and other initialization methods, concluding that there is occasional sensitivity to the initial solution and that the proposed method is effective.

Stage 1: Neighborhood Search.

Step 1b: Consider agent removal. We are given the incumbent $\mathbf{x}$ and a move size q. Consider the set of solutions obtained by removing q agents of a single type; denote it $\mathcal{X}_1(\mathbf{x}, q) = \{\mathbf{y} : \mathbf{y} = \mathbf{x} - q\mathbf{e}_i, x_i \geq q\}$ and call it a *remove neighborhood*. These solutions are evaluated; if at least one is E-feasible, then the new incumbent is the one minimizing the ratio of global-SL decrease to cost decrease, where the global-SL decrease is estimated by the evaluator; otherwise, we have determined that $\mathcal{X}_1(\mathbf{x}, q)$ contains no E-feasible solutions. Step 1b is implemented as function `Remove` in Online Appendix A.1.

Step 1c: Consider agent switching. We are given the incumbent $\mathbf{x}$ and a move size q. We select an agent type i to be reduced, called *pivot*, via a rule specified below. Consider the set of solutions obtained by decreasing x_i by q and increasing the number of agents of a less-expensive type by q; denote it $\mathcal{X}_2(\mathbf{x}, q, i) = \{\mathbf{y} : \mathbf{y} = \mathbf{x} - q\mathbf{e}_i + q\mathbf{e}_j,\ x_i \geq q,\ c_j < c_i\}$ and call it a *cost-reducing switch neighborhood*. These solutions are evaluated; if at least one is E-feasible, then the new incumbent is set by the same minimization criterion as during agent removal; otherwise, we have determined that all elements of $\mathcal{X}_2(\mathbf{x}, q, i)$ are E-infeasible. To explain the pivot selection rule, suppose we were to consider all possible pivots; then in the worst case we would have to evaluate $O(m^2)$ neighbors for the typical incumbent, which may be prohibitive. (This calculation assumes that for the typical incumbent, there are $O(m)$ possible pivots and for each of these pivots there are $O(m)$ candidates to increment.) Indeed, considering all pivots led to unacceptably large work in our Example 2, in Section 5.2, where $m = 89$. The set of candidate pivots is $\mathcal{P} = \{i : x_i \geq q,\ q_i^* > q\}$, where q_i^* is the smallest q such that all elements of $\mathcal{X}_2(\mathbf{x}, q, i)$ are known (from previous steps) to be E-infeasible; this is justified in Proposition 4 below. The pivot is selected randomly, uniformly over $\mathcal{P}$. Step 1c is implemented as function `Switch` in Online Appendix A.1.

Stage 1 termination and move size selection. We define *normal termination* (of stage 1) to mean that the incumbent $\mathbf{x}$ is *locally optimal* in the sense that $\mathcal{X}_1(\mathbf{x}, 1)$ and $\cup_{i:x_i \geq 1} \mathcal{X}_2(\mathbf{x}, 1, i)$ contain no E-feasible solutions. In words, every possible removal of one agent and every possible cost-reducing switch between two agents is deemed infeasible. Otherwise, we have *early termination*; this happens

when a work (CPU time) limit is reached before normal termination occurs. The move size q in step 1a is a positive integer that is no larger than $\max_i x_i$ and is equal to 1 with positive probability. Online Appendix A.3 provides an experimental assessment of different move-size selection rules (both deterministic and random) and finds little sensitivity.

Stage 2: Simulation-based adjustment. The solution after Stage 1 may be infeasible or suboptimal as a consequence of evaluator error. Thus, we turn to simulation as the evaluator and use local search to correct infeasibility and/or further reduce the cost, as explained below. By design, only few solutions are examined. Below, $\hat{g}_j$ is the estimated class-j service level and $\hat{f}_{i,j}$ is the estimated rate of type-j service completions at station i.

Stage 2a. The first thing we do is to simulate the incumbent of Stage 1. If the incumbent has class-specific constraint violations, then these are first addressed. The main steps are: find the class j^* with maximum violation; find the agent type i^* whose fraction of busy time spent serving class j^* is maximum, i.e., $i^* = \arg\max_{i:x_i>0}(\hat{f}_{i,j^*}/\mu_{i,j^*})/\sum_{j\in\mathcal{S}_i}(\hat{f}_{i,j}/\mu_{i,j})$; and add one agent of this type. This is continued until the constraints for all classes are satisfied. If the resulting incumbent violates the global constraint, then we iteratively add one agent of the type that maximizes the occupancy-to-cost ratio, until this constraint is satisfied. This yields a SIM-feasible solution. This is implemented as Procedure `SIMAdd` in Online Appendix A.1.

Stage 2b. We seek to reduce cost subject to maintaining SIM-feasibility, considering only single-agent removals. We maintain a list of agent types that are candidates for removal. While the list is non-empty, we: (1) calculate a measure of "excess capacity" for each agent type in the list: $\chi_i = \sum_{j\in\mathcal{S}_i} w_{i,j}(\hat{g}_j - l_j)$, where $w_{i,j}$ is the estimated fraction of type-i agents' busy time that is spent serving type-j calls; (2) sort the list by decreasing value of $\chi_i c_i$; (3) simulate the solution obtained by removing one agent of the type at the top of the list; if it is SIM-feasible, then set it as the new incumbent and reconsider the entire list (i.e., continue the While statement above); otherwise, remove this type from the list and repeat step 3 above. This is implemented as Procedure `SIMRemove` in Online Appendix A.1.

The algorithm description is complete. We now establish results on the search and discuss

algorithm enhancements.

Properties of neighborhood search. Write $\tilde{g}_\bullet$ for the evaluator's estimates of the service-level functions $g_\bullet$. For any given solution $\mathbf{x}$, we consider the condition

$$[\tilde{g}_j(\mathbf{x} - q_1\mathbf{e}_i + q_1\mathbf{e}_k) < \tilde{g}_j(\mathbf{x})] \;\Rightarrow\; [\tilde{g}_j(\mathbf{x} - q_2\mathbf{e}_i + q_2\mathbf{e}_k) \leq \tilde{g}_j(\mathbf{x} - q_1\mathbf{e}_i + q_1\mathbf{e}_k) \text{ for all } q_2 > q_1] \quad (12)$$

for all j and for all i and k with $c_i > c_k$. In words, this says that if some approximate service level $\tilde{g}_j$ decreases after a switch of size q_1, then it decreases by at least as much for all larger switch sizes. Condition (12) does not always hold in general for arbitrary solutions. For instance, it is possible to construct examples where $\tilde{g}_j(\mathbf{x} - q\mathbf{e}_i + q\mathbf{e}_k)$ is U-shaped as a function of q for some j, in which case the condition fails. However, these examples are not typical of a well-behaved call center.

Proposition 4 *1. Normal termination of stage 1 occurs after a finite number of evaluations.*

2. Suppose that (12) holds for the incumbent solution $\mathbf{x}$ *after normal termination. Then* $\cup_{q\geq 1}\cup_{i:x_i\geq q}$ $\mathcal{X}_2(\mathbf{x}, q, i)$ *contains no E-feasible solutions.*

Proof. Write $x_i^{(1)}$ for the i-th component of the initial solution. Since only cost-reducing moves are accepted, the possible incumbents are contained in $\mathcal{K} = \{\mathbf{x} : \mathbf{x} \text{ integer-valued } m\text{-vector}, \sum_{i=1}^m c_i x_i < c^{(1)}\}$, where $c^{(1)} = \sum_{i=1}^m c_1 x_i^{(1)}$; and an element of $\mathcal{K}$ can become incumbent at most once. Thus, the number of incumbents is at most $|\mathcal{K}|$. For each incumbent, there are at most $\sum_{i=1}^m x_i^{(1)} = \tilde{q}$ possible move sizes; and for each incumbent and move size, there are at most m possible removals and at most m^2 possible switches. Thus, stage 1 requires at most $|\mathcal{K}|\tilde{q}(m + m^2)$ evaluations. In the special case where $q = 1$, this bound improves to $|\mathcal{K}|(m + m^2)$. To prove part 2, observe that normal termination implies that $q_i^* = 1$ for all i with $x_i \geq 1$. In view of (12), condition $q_i^* \leq q$ implies that $\mathcal{X}_2(\mathbf{x}, q, i)$ contains no E-feasible solutions for each i, and the result follows. □

Multistart. We run (start) the algorithm several times, each with different initial solution, and retain the cost-minimal solution. Because of simulation noise, each run yields a solution that has small positive probability of being infeasible (despite being SIM-feasible). As the number of runs increases, the retained solution is more likely to be infeasible (because of selection bias). This suggests avoiding an excessively large number of runs.

Work allocation to stages and runs. We control stage-1 work via a CPU time limit. Stage 2 work is well modeled as $\kappa_2 T(N_{\mathrm{A}} + N_{\mathrm{R}})$, where T is the number of simulated hours per solution; N_{A} and N_{R} are the number of solutions simulated in stage 2a and 2b, respectively; and κ_2 is the work per simulation of 1 hour of operation. We found empirically *conservative* estimates $E[N_{\mathrm{A}}] \leq 3\sqrt{\rho}$ and $E[N_{\mathrm{R}}] \leq 3\sqrt{\rho} + m$, where the *aggregate load* is $\rho = \sum_{j \in \mathcal{N}} \rho_j$ and $\rho_j = \lambda_j/\mu_j$ is the *class-j load*, where μ_j is a station-independent class-j service rate (the formula would need adjustment otherwise). These estimates and knowledge of κ_2 allow roughly controlling stage-2 work via T. In multistart, an *even-split* rule is simple and reasonable: split the remaining work budget evenly across starts and across stages, i.e., for each run $i = 1, 2, \ldots, k$, allocate to stage 1 the fraction $1/[2(k-i+1)]$ of the budget, then allocate to stage 2 the fraction $1/[2(k-i)+1]$ of the budget.

The approach generalizes easily to formulations with constraints on performance measures other than service level, as long as the evaluator provides reasonable estimates. See Online Appendix A.0 for the LD approximation of abandonment fractions and mean waiting times for each call class.

5 Numerical comparison to the cutting-plane-and-simulation approach

To solve (P2) in its generality, i.e., with multiple constraints, the only method we know is that of Cezik and L'Ecuyer (2008) (CP). We therefore compare our approach (RS) to CP. We discuss in detail two examples that arose in collaboration with Bell Canada. We also experimented with other examples, but the ones we discuss summarize adequately our findings.

Assessing solution feasibility and algorithm performance. We assess algorithm performance over a wide range of algorithm *work* (CPU time). Work is controlled by the number of simulated hours of operation T (beyond a warm-up period). We remarked that both approaches deliver infeasible solutions with non-negligible frequency, even under a large work budget. To assess solution quality in light of this, we do a number of independent runs with each approach, and check the final solution's feasibility by a simulation that is more accurate than during optimization ($T = 12800$ hours of operation in apparent steady-state). The *empirical optimum* is the lowest-cost solution found, across runs and the two approaches, that passes this feasibility test. In some cases, we made

small manual corrections (adding one or two agents) to get feasibility for a nearly-feasible low-cost solution. The true optimum is unknown. For each approach, we report: (a) the minimum and median cost of the feasible solutions only; the number of runs for which the solution is: (b) feasible and within 1% of the empirical optimum (P_1^*); (c) within 1% of this cost, regardless of feasibility (P_1); (d) feasible, regardless of cost (P^*); and (e) the average maximum relative constraint violation in percent, $\bar{G}$, i.e., the average of $100\max\{[l-g(\mathbf{x}^*)]/l,\ [l_{j^*}-g_{j^*}(\mathbf{x}^*)]/l_{j^*}\}$ conditional on the solution $\mathbf{x}^*$ being infeasible, where $j^* = \arg\max_{j\in\mathcal{N}}[l_j - g_j(\mathbf{x}^*)]$ is the *critical class*. Infeasibility of delivered solutions is reported in Atlason et al. (2008); Green et al. (2001); Cezik and L'Ecuyer (2008); to our knowledge, our study is the first to measure the frequency and size of the infeasibility.

We now summarize call center parameters, algorithm implementation, and general behavior that apply to all examples.

Call center parameters. Customer patience is exponential. In the absense of reliable patience estimates, we consider two highly different cases for the rate: $\eta = 20$ per hour (abandonment) and $\eta = 0.02$ (very low abandonment). The cost of an agent with $s+1$ skills is $1 + 0.05s$. We have a global SL target $l = 0.80$ and acceptable waiting time $\tau_j = \tau = 20$ seconds.

CP *implementation.* We used parameter values suggested in Cezik and L'Ecuyer (2008); see Online Appendix A.5. Solution quality was sometimes sensitive to parameters and fine-tuning these is beyond our scope. In our large problem, solving the integer program (IP) to optimality required work that was often excessive. Thus, we consider two variants: (i) solve the IP exactly (CP_{IP}); and (ii) solve the linear programming (LP) relaxation and then round up each variable in the final solution (CP_{LP}). We did not use multistart with CP because the work per start is generally high. One referee suggested a third possibility: (iii) solve the IP, but not all the way to optimality; stop as soon as the relative duality gap goes below a given threshold, such as 1% or 0.5%, for example. This approach could appear as a good competitor to (ii) when the budget is too small to apply (i), because it is likely to provide a better solution to the IP problem than just rounding up the LP solution. However, our experiments with it were somewhat disappointing, especially for small work budgets. For (i) and (ii), the optimal cost increases monotonously with the iterations, as we add new constraints. But for (iii) this is no longer true and (according to our empirical observations)

it tends to take significantly more iterations on average to converge to a feasible solution of the sample problem. This number of iterations also tends to have larger variance. Moreover, solving the IP in (iii) requires significantly more work than solving the LP unless we are ready to accept a large duality gap. All of this means shorter simulation lengths at each iteration, i.e., a smaller sample size for the sample problem, for a given total work budget. This in turns gives a more noisy sample problem, whose constraints are likely to have more areas of non-concavity (increasing the chances of bad cuts in the CP method), and whose optimal solution also tends to be farther from the optimal solution of the exact problem. Approach (iii) performed more poorly than (ii) on our large example; it tended to return solutions with large infeasibility gaps $\bar{G}$ on average. For this reason, we do not report the detailed results with this method.

RS *implementation.* Labels CC1A and CC1L will refer to Example 1 with moderate and very low abandonment, respectively, and use of the LDA and LDN approximation, respectively; and likewise for Example 2. Our experience is that LDN dominated LDA in problems with very low abandonment, in the sense that it led to better solutions for similar work or faster execution for similar solution quality. The approximation accuracy is $\epsilon = 10^{-4}$ and $k_{\mathrm{U}} = 400$. In the LDA variant, we set $\psi = 2$. Requiring higher accuracy or increasing the queue size in LDA (via ψ) did not produce noticeable differences. Multistart was applied; the initial solution was constructed with $\xi = 0.8$ (the global SL target) and a different β in each start; the number of starts was increased ad-hoc with the work budget. The move size was $\max(1, \mathrm{round}(X))$, where X is an exponential random variable with mean equal to the median of the incumbent's elements. Work allocation to starts and stages followed the even-split rule and was such that total work is comparable to CP.

Effect of early termination. In side experiments, early termination had a negative effect on final cost. In CC2A, normal termination gave a median gap to empirical optimum of about 2.4%, while limiting the stage-1 work to 10% of the average work to normal termination led to a gap of 5.3%. A similar but weaker effect was present in CC2L. One remedy is to speed up the LD approximation by requiring lower accuracy. This means that for a fixed amount of time allocated to Stage 1, more solutions are examined and normal termination is more likely to occur. We did that for CC2A.

Computing platform and tools. All experiments were done on a 2.0GHz AMD Opteron processor

running Linux; we used SUN Java Development Toolkit, version 1.4.2. Linear and integer programs were solved by CPLEX, version 9.0. Our call-center simulator is likely to be much faster than typical (e.g., commercial) simulators (Buist and L'Ecuyer, 2005); thus, our comparison favors CP because this approach is more simulation-intensive than ours.

5.1 Example 1: a medium-size center

The following example is based on discussions with our industrial partner. We use different minimal service levels per call type for illustrative purposes. There are 7 call classes and 10 agent types, each having 1 or 2 skills. Overflow routing is acyclic and the data are: $\mathcal{R}_1 = \{1\}, \mathcal{R}_2 = \{1, 3\}, \mathcal{R}_3 = \{2, 4\}, \mathcal{R}_4 = \{5, 4, 3, 6\}, \mathcal{R}_5 = \{7, 6, 8, 9\}, \mathcal{R}_6 = \{9\}$ and $\mathcal{R}_7 = \{10, 8\}$. Agent type 1 prioritizes class 1 over 2. Agent type 3 prioritizes class 2 over 4. Except when priority applies, calls are served in the order of their arrival (FIFO). We have target service levels $(l_j)_{j=1}^7 = (.80, .80, .80, .75, .60, .60, .60)$, arrival rates $(\lambda_j)_{j=1}^7 = (200, 133, 323, 760, 95, 10, 380)$, and service rates varying by call class only: $(\mu_j)_{j=1}^7 = (7.7, 7.7, 7.5, 7.7, 15, 7.7, 15)$. The aggregate load is 218. The empirical optimal costs for examples CC1L and CC1A are 241.30 and 222.65, respectively.

Stage 1 empirical data and algorithm parameters. Stage 1 work for a single start averaged a few seconds. Early termination was unnecessary. At the end of Stage 1, the approximation usually overestimated the SL. In the low-abandonment case (CC1L), this solution did not require much adjustment and its cost usually differed by less than 2% from the final cost. In the other case (CC1A), the stage-1 cost was about 10% lower than the final cost, and a much bigger adjustment was necessary in stage 2. Multistart was applied with $\beta \in \{0.2, 0.5, 0.7, 0.9\}$.

Comparison to CP. Table 1 contains results for CC1A. We omit results for intermediate work budgets because they tended to interpolate the presented ones and did not reveal additional information. We see that infeasibility occurs with non-negligible frequency, and this persists up to our largest work budget. However, the expected constraint violation conditional on infeasibility, $\bar{G}$, is small and decreases steadily with work. In view of this, we declare a solution (obtained in a single run) as "good" if it is within 1% of the empirical optimum, regardless of feasibility. The main result is: as the work budget becomes smaller, RS delivers a good solution more frequently than

Case	Algo.	T	CPU_{avg}	Min. cost	Med. cost	P_1^*	P_1	P^*	$\bar{G}$
1	RS	25	55s	224.40	224.85	2	30	4	1.7
	CP_{IP}	25	2m15s	222.95	225.75	3	16	11	3.0
	CP_{LP}	25	30s	225.05	227.08	0	14	6	2.6
4	RS	640	17m10s	223.20	224.10	19	27	22	0.3
	CP_{IP}	640	15m29s	223.25	224.58	8	21	14	0.5
	CP_{LP}	1280	17m26s	223.05	224.70	14	22	23	0.6
5	RS	1920	54m02s	223.25	224.40	15	23	24	0.2
	CP_{IP}	2560	59m12s	223.00	224.05	22	31	23	0.2
	CP_{LP}	3840	57m30s	223.85	224.85	12	17	24	0.3

Table 1: Problem CC1A: Comparison of RS to CP based on 32 runs. CPU_{avg} is the average CPU time per run, in minutes (m) and seconds (s).

CP (higher P_1 values seen in cases 1 and 4). Performance differences become smaller as the budget increases (case 5). Staffing solutions occasionally differed substantially between the approaches. In the low-abandonment problem CC1L, performance differences were smaller, but our approach showed again an advantage under smaller budgets (detailed results omitted). In the empirical optimum, the fraction of calls that abandon was about 5.5% in CC1A and 0.03% in CC1L.

5.2 Example 2: a large center

This example was provided by our industrial partner about three years ago. They gave us the call types, the skill sets, and the routing rules. For each call class, they also provided the number of calls that arrived and the aggregate call handling time over a short period of time. From this, we *estimated* the (class-specific) mean service times; the arrival rates were then rescaled so that the aggregate load is 500. Exactly the same example was used by Ceşik and L'Ecuyer (2008). Of all problems we tried, this was the most difficult. The complete data for this example is available at `http://www.iro.umontreal.ca/~lecuyer/papers.html`, next to the entry of this paper.

There are 65 call classes and 89 agent types. The arrival rates vary from 1.046 to 416.6. Except for 9 classes whose aggregate load is below 3, the service rates are between 4.32 and 12.79. This is a virtual call center with two distinct physical locations. Calls are distinguished by location and needed skill, so for a given skill there are two call classes, one for each location. Frequently needed skills are found at both locations. Location one has 22 call classes and 15 agent types; location

two location has 43 call classes and 74 agent types. The number of skills per agent ranges from 1 to 24. Upon call arrival, a call may be immediately assigned only to a local agent. If no local agent is available, then the call is placed in a local queue; as soon as the call has spent 6 seconds in queue, the ACD tries to assign it again, this time considering both local and remote agents, and preferring local ones. In the routing list, local agents precede remote ones (this induces cycles in the flow graph); within each location, lower number of agent skills comes first, and ties are broken arbitrarily. Of all agents of the same type, the individual agent selected is the one with longest idle time. Whenever an agent becomes free, she gives priority to the local queue. We set a service level target of 50% per class to reflect that the company usually wants to rule out solutions in which certain classes receive very poor service. In another experiment with a target of 80% per class, the relative performance of the methods was roughly the same.

Stage 1 empirical data and algorithm parameters. Stage 1 work varied considerably and averaged 440 seconds for CC2L and 540 seconds for CC2A. Early termination was necessary on several runs for CC2A. Multistart was applied with $\beta \in \{0.6, 0.8\}$ except for the largest work budget, where this set is $\{0.2, 0.5, 0.6, 0.7, 0.8, 0.9\}$. The cost gaps between the solutions after Stage 1 and Stage 2 are similar to those in Example 1.

Comparison to CP. Table 2 contains the main results. RS* denotes reduced LD accuracy ($\epsilon = 10^{-3}$). The work budgets are larger than Example 1 because the output is noisier. A clear pattern emerges: our approach yields lower-cost solutions than CP for all work budgets except the largest one, where the two are comparable. In some cases, the cost margin is large. The underperformance of CP is a consequence of simulation noise that is too large, in this example. With CP, a very bad cut was occasionally seen: for CC2A with 27 minutes average work, one run gave a cost of 975.05 while the 15 others ranged from 617.80 to 643.25 (across all runs, some of which gave infeasible solutions).

Table 3 gives more information on the empirical optimum and typical solutions. In CC2L, the CP solution has a large violation for one constraint. Our best solutions in CC2L have substantial slack on the global SL constraint; the staffing is dictated by a constraint for a single class whose load is usually low. In the empirical optimum, the fraction of calls that abandon was about 7.5%

Ab	Case	Algo.	T	CPU_{avg}	Min. cost	Med. cost	P_1^*	P_1	P^*	$\bar{G}$
L	1	RS	80	24m52s	660.55	663.60	3	10	6	13.6
		CP_{LP}	25	22m51s	668.75	668.75	0	4	1	20.0
	2	RS	320	45m44s	657.95	663.00	2	13	3	5.7
		CP_{LP}	80	58m10s	677.20	677.20	0	6	1	15.7
	3	RS	1280	435m31s	657.20	659.50	9	15	9	1.9
		CP_{LP}	960	567m52s	657.35	659.45	6	14	7	2.3
A	1	RS	25	52m01s	612.65	615.05	0	1	3	7.0
		RS*	25	30m08s	608.80	613.32	0	0	6	7.4
		CP_{LP}	25	27m15s	631.10	634.65	0	0	3	8.6
	3	RS	160	96m56s	608.85	611.35	0	0	7	1.5
		RS*	160	46m12s	607.25	610.35	0	0	7	2.8
		CP_{LP}	80	60m21s	616.25	621.95	0	0	3	8.3
	4	RS	640	804m25s	606.10	608.52	0	0	8	2.4
		RS*	640	420m16s	605.65	607.68	1	4	8	1.1
		CP_{LP}	640	295m10s	605.20	613.25	1	5	5	2.0

Table 2: Example 2: Comparison of RS to CP based on 16 runs. 'A' and 'L' in column 'Ab' denote the cases of moderate and very low abandonment, respectively.

in CC2A and 0.01% in CC2L.

Ab	Case	Algo.	Cost	SL	SL_{j^*}
L	*	-	657.00	0.850 ± 0.001	0.501 ± 0.004
	2	RS	661.80	0.867 ± 0.001	$\mathbf{0.487 \pm 0.006}$
		CP_{LP}	664.95	0.868 ± 0.002	$\mathbf{0.36 \pm 0.02}$
A	*	-	600.00	0.812 ± 0.001	0.505 ± 0.003
	3	RS	610.55	0.865 ± 0.001	0.516 ± 0.003
		RS*	610.35	0.803 ± 0.001	0.500 ± 0.002
		CP_{LP}	617.70	0.842 ± 0.001	$\mathbf{0.487 \pm 0.002}$

Table 3: Example 2: Cost and service level of the empirical optimum ("*") and typical solutions. Service levels below the target are typed in bold.

6 Comparisons to existing and alternative approaches

6.1 The method of Wallace and Whitt (2005)

Modified Example 2 with skill-set constraints relaxed. We modify the example to fit the assumptions of Wallace and Whitt (2005) (WW). Each agent has exactly two skills (call types) designated primary and secondary, and each skill pair exists in each location. There are 4160 agent types, defined by the ordered pair of the two skills and the agent's physical location. The routing rule is

that of WW. The WW algorithm yielded solutions with 445 and 404 different agent types in CC2L and CC2A, respectively; in both cases, the cost was about 13% below our empirical optimum. This large cost reduction is easy to explain: some skill pairs whose load is relatively large do not exist alone; they are "bundled" with other skills; so when this agent type is needed in the skill-constrained problem, he/she is considerably costlier than in the relaxed problem.

Adapting the WW approach. All skill set constraints, cost structure, and routing are those of our original examples. The square-root staffing formula of WW was applied for initialization, subject to complying with existing skill sets: whenever they put a number of agents having a skill pair, we put the same number of the cheapest of the existing agent types having this skill pair; if the pair did not exist, then we put the cheapest agent type having the primary skill. Their algorithm was adapted to account for unequal cost among agent types. This gave solutions of much higher cost than ours in Example 2, and comparable in Example 1. In two runs of our algorithm with these solutions as initial ones, we found much better solutions, but still not as good as found by our main approach. Thus, we have no reason to believe that this adapted WW approach is competitive.

6.2 Alternative Stage 1 evaluator and alternative optimizer

Generalizing the proposed approach, one can combine some *optimizer* with some fast approximate *evaluator* of service levels (Stage 1) and then apply, if necessary, a local *adjustor* supported by a more accurate evaluator (Stage 2). We specify several possibilities and report algorithm performance for selected examples. One evaluator we consider is the loss approximation of Koole and Talim (2000). The main finding of this comparison, which we detail next, is that the final staffing appears to be rather insensitive to the errors in the underlying service-level approximation.

The loss-approximation of Koole and Talim (2000) as evaluator. We replace the LD approximation by the KT loss approximation and call the resulting staffing algorithm RS/KT. Thus, during Stage 1, we determine feasibility by comparing one minus the approximated loss rates to the corresponding target SL values. We compared RS/KT to our standard approach (RS/LD) in our four problems. For problems CC2L and CC2A, the final costs were quite comparable. The biggest cost differences occurred in CC1L: in 32 runs with parameters as in case 4, the minimum and median

RS/KT cost were 0.5 % and 0.9 % above the corresponding values of RS/LD, respectively. At the end of Stage 1, the KT error is generally higher than the LD one, especially in the low-abandonment case (note that Stage 1 of RS/KT behaves independently of the abandonment aspect of the call center). In most of the 32 runs for CC1L, the global SL of the RS/KT incumbent at the end of Stage 1 was under 10%, much lower than its typical counterpart with RS/LD. Not surprisingly, the KT error is smaller in CC1A, where behavior is closer to a loss system. As a consequence of larger SL error with KT, Stage-2 execution times were much higher.

The importance of having a somewhat good approximation is evidenced by additional experiments in which we pretended having no approximation and ran Stage 2 only, starting with one agent of each type. This worked well in Example 1; but in Example 2, the typical cost was about 10% above the empirical optimum. In summary, we find: a somewhat good approximation is essential in finding good moves during Stage 1, and this is an essential part of our approach; smaller approximation error is additionally helpful in reducing the Stage-2 execution time.

Simulation as evaluator. We replace the LD approximation by simulation and call the resulting algorithm RS/SIM. Some experimentation was needed to find appropriate simulation lengths for the Stage 1 evaluator, say T_1, and for the adjustor, say T. In general, T_1 must be small enough so that enough solutions are examined; and T must be large enough to avoid large infeasibility in the final solution. RS/SIM was ineffective in both variants of Example 2. In CC2A, with $T_1 = 25$, $T = 160$, and $\mathcal{B} = \{0.6, 0.8\}$, the costs for 5 runs ranged in [627.95, 637.15] and work was about 15 hours. Our experiments suggest that this method is unlikely to be effective, except perhaps in small dimensions, because of the large number of solutions that must be evaluated by simulation.

Alternative optimizer. Approach CP/LD combines the cutting-plane optimizer of Cezik and L'Ecuyer (2008), the LD evaluator, and the simulation-based adjustor. CP/LD may be interesting when one is willing to accept some loss in solution quality in exchange for faster execution. In problem CC2A, CP/LD reduced stage-1 work drastically relative to RS: the number of evaluations was cut by a factor of 50, and execution time was cut by a factor of 10. However, the overall speedup was limited because of the need for simulation-based adjustment.

7 Conclusion

We formulated the problem of staffing in a multi-skill call center as a mathematical program (P2) with constraints on the service levels. We developed a solution approach using search methods that are supported by the loss-delay approximation of class-specific service levels. The approximation is most relevant when the routing policy belongs to the overflow-routing family. The search delivers a staffing that is locally optimal with respect to the approximate service-level functions. This solution is then adjusted for either feasibility or cost reduction via simple local search methods, where simulation provides unbiased (noisy) estimates of service levels.

We compared our approach to the only practical alternative we know (Cezik and L'Ecuyer, 2008). Comparison was via examples for which (partial) data were provided by our industrial sponsor, including explicit constraints on skill sets and policies in the overflow-routing family. We assumed Poisson arrivals and exponential service times and considered both substantial and low (exponential) abandonment. We solved problems for a wide range of work budget. Our approach usually delivered better solutions than the alternative, and this advantage increased as the work budget decreased. It also appears that the advantage increases as problem dimension increases.

Crucial in our approach is a fast approximation that selects good moves among the many possible choices; the approximation's accuracy does not appear to be crucial. In our experiments, the loss-delay approximation had a small advantage over the loss approximation of Koole and Talim (2000): it gave noticeably better solutions in one example (with very high customer patience); and it led to faster execution. We conclude that our search methods supported by the loss-delay, and possibly by other approximations, can be a useful tool in multi-skill staffing.

Acknowledgments

This research has been supported by Grants OGP38816-05 and CRDPJ-320308 from NSERC-Canada, a grant from Bell Canada via the Bell University Laboratories, and a Canada Research Chair, to the third author. The second author benefited from a scholarship provided jointly by NSERC and Bell Canada. The paper was written in part while the third author was at IRISA, in

Rennes, France.

References

Aho, A. V., J. E. Hopcroft, and J. D. Ullman. 1974. *The design and analysis of computer algorithms*. Reading, MA, USA: Addison-Wesley.

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

Atlason, J., M. A. Epelman, and S. G. Henderson. 2008. Optimizing call center staffing using simulation and analytic center cutting plane methods. *Management Science* 54 (2): 295–309.

Bassamboo, A., J. M. Harrison, and A. Zeevi. 2006. Design and control of a large call center: Asymptotic analysis of an LP-based method. *Operations Research* 54 (3): 419–435.

Bhulai, S., G. Koole, and A. Pot. 2008. Simple methods for shift scheduling in multi-skill call centers. *Manufacturing and Service Operations Management* 10:411–420.

Buist, E., and P. L'Ecuyer. 2005. A Java library for simulating contact centers. In *Proceedings of the 2005 Winter Simulation Conference*, 556–565: IEEE Press.

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

Chan, W. 2006. Optimisation stochastique pour l'affectation du personnel polyvalent dans un centre d'appels téléphoniques. Master's thesis, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal, Canada.

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

Chevalier, P., R. A. Shumsky, and N. Tabordon. 2004. Routing and staffing in large call centers with specialized and fully flexible servers. Technical report, Simon Graduate School of Business, University of Rochester.

Cooper, R. B. 1981. *Introduction to queueing theory*. second ed. New York: North-Holland.

Franx, G. J., G. Koole, and A. Pot. 2006. Approximating multi-skill blocking systems by hyper-exponential decomposition. *Performance Evaluation* 63:799–824.

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

Green, L. V., P. J. Kolesar, and J. Soares. 2001. Improving the SIPP approach for staffing service systems that have cyclic demands. *Operations Research* 49 (4): 549–564.

Harrison, J. M., and A. Zeevi. 2005. A method for staffing large call centers based on stochastic fluid models. *Manufacturing and Service Operations Management* 7 (1): 20–36.

Henderson, S., and A. Mason. 1998. Rostering by iterating integer programming and simulation. In *Proceedings of the 1998 Winter Simulation Conference*, Volume 1, 677–683.

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

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

Koole, G., and J. Talim. 2000. Exponential approximation of multi-skill call centers architecture. In *Proceedings of QNETs*, 23/1–10.

Pot, A., S. Bhulai, and G. Koole. 2008. A simple staffing method for multi-skill call centers. *Manufacturing and Service Operations Management* 10:421–428.

Riordan, J. 1962. *Stochastic service systems*. New York: John Wiley & Sons Inc. The SIAM series in applied mathematics.

Ross, S. M. 1983. *Stochastic processes*. Wiley Series in Probability and Mathematical Statistics.

Wallace, R. B., and W. Whitt. 2005. A staffing algorithm for call centers with skill-based routing. *Manufacturing and Service Operations Management* 7 (4): 276–294.

Wolff, R. W. 1989. *Stochastic modeling and the theory of queues*. New York: Prentice-Hall.

Author Biographies

Athanassios N. Avramidis is Lecturer in Operational Research in the School of Mathematics at the University of Southampton, United Kingdom. This work was done while he was Researcher in the Département d' Informatique et de Recherche Opérationnelle at the Université de Montréal, Canada. He has been on the faculty at Cornell University and a consultant with SABRE Decision Technologies. His main research interests are Monte Carlo and discrete-event stochastic simulation, particularly efficiency improvement via variance reduction, and stochastic modeling in industrial and service systems. His recent research articles are available on-line from `http://www.personal.soton.ac.uk/~aa1w07`.

Wyean Chan is a MSc Student in the Département d'Informatique et de Recherche Opérationnelle, at the Université de Montréal, Canada. His main interests are object-oriented programming, applied mathematics and optimization. He is currently working on the development of staffing tools for call centers. His e-mail address is `chanwyea@IRO.UMontreal.CA`.

Pierre L'Ecuyer is Professor in the Département d'Informatique et de Recherche Opérationnelle, at the Université de Montréal, Canada. He holds the Canada Research Chair in Stochastic Simulation and Optimization. His main research interests are random number generation, quasi-Monte Carlo methods, efficiency improvement via variance reduction, sensitivity analysis and optimization of discrete-event stochastic systems, and stochastic simulation in general. He is currently Associate/Area Editor for *ACM Transactions on Modeling and Computer Simulation*, *ACM Transactions on Mathematical Software*, *Statistical Computing*, *International Transactions in Operational Research*, *The Open Applied Mathematics Journal*, and *Cryptography and Communications*. He obtained the *E. W. R. Steacie* fellowship in 1995-97, a *Killam* fellowship in 2001-03, and became an INFORMS Fellow in 2006. His recent research articles are available on-line from his web page: `<http://www.iro.umontreal.ca/~lecuyer>`.