A STAFFING ALGORITHM FOR CALL CENTERS WITH SKILL-BASED ROUTING

by

Rodney B. Wallace

IBM
rodney.wallace@us.ibm.com

Ward Whitt

Columbia University
ward.whitt@columbia.edu

August 20, 2004, Revision: July 4, 2005

Abstract

Call centers usually handle several types of calls, but it usually is not possible or cost-effective to have every agent be able to handle every type of call. Thus, the agents tend to have different skills, in different combinations. In such an environment, it is challenging to route calls effectively and determine the staff requirements. This paper addresses *both* these routing and staffing problems by exploiting limited cross-training. Consistent with the literature on flexible manufacturing, we find that minimal flexibility can provide great benefits: Simulation experiments show that when (i) the service-time distribution does not depend on the call type or the agent and (ii) each agent has only two skills, in appropriate combinations, the performance is almost as good as when each agent has all skills. We apply this flexibility property to develop an algorithm for both routing and staffing, aiming to minimize the total staff subject to per-class performance constraints. With appropriate flexibility, it suffices to use a suboptimal routing algorithm. Simulation experiments show that the overall procedure can be remarkably effective: The required staff with limited cross training can be nearly the same as if all agents had all skills. Hence the overall algorithm is nearly optimal for that scenario.

Subject classifications: Queues, applications: call centers with skill-based routing. Queues, algorithms: staffing and routing in call centers with skill-based routing. Queues, networks: call centers with skill-based routing.

Keywords: telephone call centers, customer contact centers, staffing, routing, skill-based routing, resource pooling, cross-training, flexible servers, chaining, queues, multi-server queues, multi-class queues, simulation.

1. Introduction

The purpose of this paper is to provide insights and methods to help improve the design and management of telephone *call centers* and more general customer contact centers allowing contact through other media such as email. As indicated in the review by Gans et al. (2003), further work is needed because call centers have become quite complicated.

Skill-Based Routing. Call centers usually handle several types of calls, but it usually is not possible or cost-effective to train every agent (customer service representative) to be able to handle every type of call. For example, with the globalization of many businesses, call centers often receive calls in several *different languages*. The callers and agents each may speak one or more of several possible languages, but not necessarily all of them. And, of course, it may simply not be possible for the agents to learn all the languages. But it may well be possible to find agents that can speak two or three languages, in various combinations, especially when agents in the same "virtual call center" are working in different locations.

Another classification of calls involves special *promotions*. The callers may be calling a special 800 number designated for the special promotion. Agents typically are trained to respond to inquiries about some of the promotions, but not all of them. Learning about special promotions is certainly less difficult than learning entire languages, but it tends to be prohibitive to train all agents to be able to respond to calls about all promotions, especially when there is a very short time span between the creation and the delivery of the promotion.

Specialization also naturally arises in *strategic outsourcing*, where one company turns over some of its business functions to a third-party contractor. An example is the management of the company's computer and information systems. The strategic outsourcing is usually managed by a *single-point-of-contact help desk*, which in fact is a call center. In these technical help desks, agents may only be able to help customers with some of their technical problems. The agents may have different skills, with some skills requiring extensive training.

Moreover, with outsourcing, in some cases agents may be trained to represent more than one service provider. Again, acquiring the skills to represent a few different service providers may be feasible, but it tends to be prohibitive to train all agents to represent all service providers.

Thus, frequently, the calls have different requirements and the agents have different skills. Fortunately, modern *automatic call distributors* (ACD's) have the capability of assigning calls

to agents with the appropriate skills. Thus the call-center information-and-communication-technology (ICT) equipment can allow for the generalization to multiple call types. That capability is called *skill-based routing* (SBR).

Most current call centers perform skill-based routing, and many do so remarkably well. Nevertheless, there remains a need for fundamental principles and operations-research techniques that will make it possible to better design and manage SBR call centers. The operational complexities of SBR call centers were nicely described by Garnett and Mandelbaum (2000).

Resource Pooling. We propose a two-word answer to the SBR problem: *resource pooling*. As explained by Mandelbaum and Reiman (1998), "resource pooling" can have different meanings. Within the context of an SBR call center, we use "resource pooling" to mean that with a limited amount of cross training (having agents with multiple skills, but only a few skills in appropriate combinations) and a reasonable (suboptimal) routing algorithm, a diverse SBR call center may perform nearly the same as if all agents had all skills (and routing were not an issue at all).

In other words, we find that *a little flexibility goes a long way*. That conclusion is now a fundamental principle of flexible manufacturing; e.g., see Aksin and Karaesman (2002), Hopp and Van Oyen (2003), Jordan et al. (2003), Gurumurthi and Benjaafar (2004) and references therein. (For work on stochastic networks in the same spirit, see Azar et al. (1994), Vvedenskaya et al. (1996), Turner (1996, 1998), Mitzenmacher (1996) and Mitzenmacher and Vöcking (1999).) That work shows that only limited flexibility suffices, provided that the sharing is arranged appropriately. That sharing proviso is captured by the notion of *chaining*, articulated by Jordan and Graves (1995). Our work is consistent with that large body of previous work.

Here we go further: *We apply the limited-flexibility principle to develop a full algorithm to both route and staff in an SBR call center.* We also specify the required number of telephone trunk lines. By making simulation experiments, we show that our algorithm is nearly optimal in a reasonable SBR scenario with six call types when each agent has only two skills. However, our results here are restricted to the case in which the mean service times do not depend on the call type or the agent. That restriction is reasonable for many call-center applications, but it leaves open the applicability of the approach more generally.

Organization of This Paper. Here is how the rest of this paper is organized: In Section 2 we specify our SBR call-center and the optimization problem we want to solve. In Section 3 we specify precisely how we implement skill-based routing; it is done with a static-priority scheme based on agent-skill matrices. In Section 4 we describe our experiment to investigate resource pooling in an SBR call center. There we specify the model assumptions we make in all our simulation experiments.

In Section 5 we present our staff-and-equipment provisioning algorithm. In Sections 6 and 7 we describe simulation experiments to show how that algorithm performs. These have six call types with an offered load requiring about 90 agents. The first experiment in Section 6 is for a balanced call center, like the one considered in Section 4. The second experiment in Section 7 is for a more realistic unbalanced call center.

Afterwards, in Section 8 we discuss extensions. Additional details can be found in Wallace (2004), on which this paper draws.

Related Literature. Other interesting, but quite different, approaches to staffing SBR call centers have recently been proposed by Harrison and Zeevi (2003), Basamboo et al. (2004), Armony and Mandelbaum (2004) and Chevalier et al. (2004). See these sources for additional references. Earlier work includes Perry and Nilsson (1992), Borst and Seri (2000) and Koole and Talim (2000). In using simulation to improve call-center performance, we are following longstanding practice; see Anton et al. (1999) and Brigandi et al. (1994).

2. Our SBR Call-Center

Our SBR call-center is a multi-server queueing system with C servers, K extra waiting spaces and n call types. We assume that there is a separate queue for each call type (which could of course be virtual). The pair (C, K) corresponds to having C agents and $C + K$ available telephone trunk lines. A call uses a trunk line both while it is waiting and while it is being served. We assume that the ACD can accommodate $C + K$ calls, with any calls not being served waiting (in the queue for its type). Calls arriving when all $C + K$ trunk lines are occupied are assumed to be blocked and lost. We do not consider either abandonments or retrials; they are discussed in the final Section 8 on extensions.

We say that an agent has skill k if the agent can answer calls of type k; then the agent can be assigned any class k-call. If the agent does not have skill k, then the agent cannot be assigned class-k calls. Within this framework, we will specify a *routing policy*. A routing policy

specifies which agent should answer each call. We assume that calls are answered immediately when the assignment is made. Thus all assignments are made either when a new call arrives or when an agent completes a call and becomes free. We assume that the agent completes the call without interruption, once it has been answered.

Within this framework, the *staffing problem* is to choose the total number of agents C and specify their skills. The more general *provisioning problem* includes specifying K as well. For agents with specified skills, our goal is to lexicographically minimize (C, K) subject to per-class performance constraints. By "lexicographically minimize," we mean: first, minimize C and then, for given C, minimize K. We consider C and K lexicographically because the agents tend to be much more costly than trunk lines.

There also are costs for giving agents skills, but we do not consider those costs directly here. Instead, we treat the skills as part of the constraints. In particular, we will allow each agent to have only two skills. For this initial study, we allow total freedom in the choice of the two skills.

The specific performance constraints we consider are *speed-to-answer service-level constraints* and *blocking-probability constraints*. To state these constraints, let W_k be the steady-state waiting time before beginning service and let Q_k be the steady-state number of customers in the system, experienced by an arrival of type k. The per-class speed-to-answer service-level constraints specify that the *conditional* probability that the steady-state waiting time for type-k calls is below a desired target, given that the call is not blocked, should be above some threshold, i.e.,

$$P(W_k \leq \tau_k | Q_k < C + K) \geq \delta_k, \quad 1 \leq k \leq n \ , \tag{2.1}$$

where τ_k is the type-k target and δ_k is the type-k threshold. The per-class blocking probability constraints are

$$P(Q_k = C + K) \leq \epsilon_k, \quad 1 \leq k \leq n \ . \tag{2.2}$$

We will show that in a reasonable scenario, including the constraints in (2.1) and (2.2), we are able to obtain a near-optimal solution, even though we do not perform an elaborate optimization for the routing. Amazingly, limited flexibility and a reasonable routing policy are enough. However, we consider only two specific scenarios. Thus we only establish a *proof of concept*: We show that such near-optimality may possibly be achieved in other scenarios. Nevertheless, *we believe our results can radically change the approach to staffing and routing in SBR call centers*. Our results suggest paying more attention to obtaining the required limited

cross training, while paying less attention to the routing.

3. A Static-Priority Routing Scheme

We use a static-priority scheme for routing. As indicated before, agents have *skills*. An agent can answer any type-k call if and only if that agent has skill k. Thus an agent can have from 1 to n skills. (We are thus specifying the routing algorithm for a general case, even though we are contending that it might be sufficient for each agent to have only two skills.) Agents not only have skills, but they also have *priority levels* for those skills.

We specify the skills of agents and the priority levels of those skills via a $C \times n$ *agent-skill matrix* A. The rows of A correspond to the agents, the columns of A correspond to *priority levels* and positive entries identify skills: If $A_{i,j} = k$, then agent i has skill k at priority level j; if $A_{i,j} = 0$, then agent i has no skill at priority level j; if there is no j for which $A_{i,j} = k$, then agent i does not have skill k. Thus row i of A specifies the skills and priority levels for the i^{th} agent. Lower skill-level numbers have preference in the priority scheme. We assume that each agent has at most one skill at any given priority level, so that the matrix A can indeed be used to assign skills and priority levels to agents. Without loss of generality, we assume that each agent has each skill at only one priority level (a positive integer appears at most once in each row).

The first column of A specifies the *primary skills*, by which we mean those skills with the highest priority (lowest priority number). Thus $A_{i,1}$ is the primary skill of agent i. We assume that every agent has a primary skill, so that $A_{i,1}$ is an integer with $1 \le A_{i,1} \le n$ for each i. We group the agents by their primary skills, so that *work group* G_k is the subset of all agents with primary skill k, i.e.,

$$G_k \equiv \{i : 1 \le i \le C, \quad A_{i,1} = k\}, \quad 1 \le k \le n \ .$$

By our assumptions, the collection of work groups is a partition of the set of all agents, i.e., of the set $\{1, \ldots, C\}$. Each agent belongs to one and only one work group. The number of agents in work group k, denoted by C_k, is the number of elements in G_k, denoted by $|G_k|$, i.e.,

$$C_k \equiv |G_k| = \sum_{i=1}^{C} 1_{\{A_{i,1}=k\}}(i) \ ,$$

where 1_B is the indicator function of the set B; i.e., $1_B(i) = 1$ if $i \in B$ and $1_B = 0$ otherwise. We allow work groups to be empty, but we require that, for all k, $1 \le k \le n$, there exists at least one agent with skill k. Otherwise, class-k calls would never be answered.

In order to clarify the rules for assigning skills to agents, we consider the following four examples:

$$\mathbf{A_{5\times 1}} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \ \mathbf{A_{3\times 2}} = \begin{pmatrix} 1 & 0 \\ 2 & 0 \\ 2 & 0 \end{pmatrix}, \ \mathbf{A_{4\times 2}} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 2 & 1 \\ 2 & 1 \end{pmatrix}, \ \mathbf{A_{6\times 4}} = \begin{pmatrix} 3 & 4 & 1 & 0 \\ 1 & 4 & 0 & 0 \\ 2 & 3 & 0 & 0 \\ 4 & 0 & 0 & 0 \\ 3 & 1 & 2 & 4 \\ 1 & 0 & 4 & 0 \end{pmatrix}$$

The first matrix $\mathbf{A_{5\times 1}}$ specifies an agent profile for a call center manned by 5 agents, all possessing the same single skill. The second matrix $\mathbf{A_{3\times 2}}$ specifies an agent profile for a call center with 3 agents, each possessing one of two primary skills. The zeros in the second column indicate that no agent has a secondary skill. Thus, this call center has two separate work groups that function as separate call centers: Agent 1 will handle all type-1 calls, while agents 2 and 3 will handle all type-2 calls.

The third matrix $\mathbf{A_{4\times 2}}$ is in the spirit of the bilingual call-center model used in Green (1985) and Stanford and Grassmann (1993, 2000). In this matrix, the first two rows represent 2 agents each with a primary skill to support call type 1 and no secondary skill. These 2 agents form work group one and are referred to as the limited-use or restricted-use servers in Green (1985) and the unilingual group in Stanford and Grassmann (2000). The third and fourth rows represent agents 3 and 4. These agents each have a primary skill to support call type 2 and a secondary skill to support call type 1. They make up work group two and are referred to as the general-use servers and bilingual group in Green (1985) and Stanford and Grassmann (2000), respectively.

In the last example, the matrix $\mathbf{A_{6\times 4}}$ illustrates the more general structure possible for the agent-skill matrix. This example has 6 agents and 4 call types. There are four work groups, one for each call type. Using the first column, we can identify the agent's work group. Work group 1 consists of agents 2 and 6, while work group 2 consists of agent 3. All agents have secondary skills with the exception of agents 4 and 6. Agent 4 supports only call type 4, while agent 5 is a *universal agent*, because he can support all of the call types. However, agent 5 not only can support all call types; he does so in a specified priority order. Agent 6 can support 2 different service requests: call type 1 at the primary level and call type 4 at the tertiary level. Note that agent 6 has a skill at priority level 3 but no skill at priority level 2.

What to do when an Arrival Occurs. In order to implement skill-based routing, we need to specify the decisions we will make in two situations: (i) when an arrival occurs, and (ii)

when an agent becomes free. We treat each in turn.

Arriving calls of type k are first routed to available agents in work group k, because those agents have primary skill k ($G_k = \{i : A_{i,1} = k\}$). We use the *longest-idle-agent-routing* (LIAR) *policy* to determine which of the idle agents in work group k is to handle the call. The LIAR policy sends the call to the agent in work group k that has been idle the longest since the completion of their last job. The LIAR policy is deemed fair, because it tends to balance the workload across agents. However, other tie-breaking schemes could be used instead. For example, we could instead choose the idle agent whose cumulative idle time over the last half hour is greatest. Under our assumptions, the tie-breaking rule has no impact upon the performance measures we consider here, but the issue is nevertheless important in practice.

If all agents with call type k as a primary skill are busy, then the call is routed to available agents having call type k as a secondary skill (at priority level 2). Again, among all agents having type k as a secondary skill, the LIAR policy is used to pick the actual agent to handle the call. If all agents with call type k as a primary skill or a secondary skill are busy, then the call is routed to available agents having call type k as a tertiary skill (at priority level 3), and so on. Again, the LIAR policy is used to break ties.

If no available qualified agent can be found to handle the type-k call immediately upon arrival, then the type-k call is placed at the end of the queue associated with call type k.

What to do when an Agent becomes Free. We now specify what an agent does when he completes handling a call and becomes free. First, if there are no customers in the n queues, then the agent goes idle. Otherwise, the agent visits the queues of the call types for which he has skills. If there are no waiting calls for which he has skills, then again the agent goes idle. The agent visits the queues in order of the agent's priority levels; i.e., the agent goes first to the queue with his primary skill, second to the queue with his secondary skill, and so forth. The agent serves the call that is first in line in the first nonempty queue in the priority-level order. Thus the agent serves calls within each queue in a *first-come first-served* (FCFS) order.

4. The Resource Pooling Experiment

In this section we describe a simulation experiment conducted to investigate the extent to which resource pooling holds in SBR call centers. In particular, we investigate how many skills agents need in order for the performance to be nearly the same as if all agents had all skills.

We start by specifying the modelling assumptions we make for all our simulation experiments.

Model Assumptions. We assume that n types of calls arrive at the call center according to n mutually independent Poisson processes with rates λ_k, $1 \le k \le n$. That is equivalent to assuming that all calls arrive according to a single Poisson process with rate $\lambda = \lambda_1 + \cdots + \lambda_n$ and, afterwards, are assigned to be type k with probability $p_k \equiv \lambda_k/\lambda$, according to mutually independent trials, independent of the single Poisson process.

We assume that the call holding (service) times are mutually independent exponential random variables, independent of the arrival process, *with a common mean* $1/\mu$. We believe that this is our most restrictive assumption. It is reasonable in many applications, but if it is not nearly satisfied, then the approach in this paper may not perform well.

We do not consider either abandonments or retrials. Thus, calls that are blocked do not affect future arrivals. We are anticipating that service quality will be sufficiently high that abandonments can be ignored. Thus we assume that all admitted calls will eventually be served.

The Resource-Pooling Experiment. Under our assumptions so far, that universal-agent case is approximately equivalent to the $M/M/C/K$ model with the FCFS discipline, in which there is a single call type and a single queue. However, even when all calls have the same service-time distribution, as we are assuming, these systems are not quite equivalent, because the FCFS discipline associated with the $M/M/C/K$ model is not operating overall in our SBR system (even though it is within each queue).

We consider a call center serving $n = 6$ call types and see what happens when all agents possess m skills, allowing m to range from 1 to 6 in separate simulation runs. The model we consider is a *balanced* $M_6/M/90/30$ SBR call center: There are $C = 90$ agents and $K = 30$ extra waiting spaces. Since there are 6 call types, there are 6 work groups, each with $C_k = 90/6 = 15$ agents. The service times are IID exponential random variables with mean $1/\mu_k = 1/\mu = 10$ minutes.

Since the number of agents is 90, each work group can have exactly 15 agents, and, when $m \ge 2$, there will be exactly $15/5 = 3$ agents with each of the $6 \times 5 = 30$ combinations of the possible primary and secondary skills. When $m > 2$, we do not try hard to optimally balance the skills at lower priority levels. Instead, we let each successive skill beyond the skill assigned at priority level 2 be the next available skill. For example, suppose that, when $m = 2$,

row i of the agent-skill matrix A is $(5, 3, 0, 0, 0, 0)$. When we increase the number of skills per agent, row i is changed successively to $(5, 3, 4, 0, 0, 0)$, $(5, 3, 4, 6, 0, 0)$, $(5, 3, 4, 6, 1, 0)$ and $(5, 3, 4, 6, 1, 2)$. This procedure guarantees that each skill appears the same number of times at each priority level. This procedure also yields appropriate sharing, but evidently not optimal sharing.

However, the full procedure above does not matter much, because we are primarily interested in comparing the cases $m = 1$, $m = 2$ and $m = 6$. When $m = 6$, all agents have all skills, and the call center behaves much like a single-group call center. In contrast, when $m = 1$, the call center behaves much like 6 separate call centers, for which the performance is much worse. When $m = 1$, the call center does not behave *exactly* like separate call centers because of the finite waiting room. Here, when $m = 1$, the six call types share the common waiting room. The shared waiting room leads to much better performance than if the waiting room too were divided into segregated portions without any sharing.

Here is the main point: We show that the performance for $2 \leq m \leq 5$ is nearly the same as for $m = 6$, being much better than when $m = 1$. We thus see that significant resource pooling occurs even when each agent has only two skills. The performance is somewhat better if $m = 3$ than if $m = 2$, but most of the resource-pooling benefit occurs when $m = 2$.

Recall that the offered load with common mean service times is the arrival rate times the mean service time. We consider three different offered loads: 84.0 (normal load), 77.4 (light load) and 90.0 (heavy load). The corresponding traffic intensities are: $84.0/90 = 0.933$ (normal load), $77.4/90 = 0.86$ (light load) and $90/90 = 1.00$ (heavy load). As discussed in Whitt (1992), the interpretation of traffic intensities depends on the number of servers, with the normal load increasing as the number of servers increases. The finite waiting room makes it possible to have traffic intensities greater than 1 (as would customer abandonment, which we are not considering).

In our resource-pooling simulation experiment, we examine all possible cases, considering each number of skills with each loading. Thus we perform $6 \times 3 = 18$ simulation runs in all. Each simulation run is based on approximately $800,000$ arrivals, starting after an initial warmup period to allow the system to reach steady state. The warmup period was chosen to correspond to 2000 mean service times, which constituted about 20%–24% of each run. In order to calculate confidence intervals, we used the technique of batch means, dividing each run into 20 batches.

It is important to validate the simulation tool. As described in Chapter 4 of Wallace (2004),

the simulation tool was validated by making comparisons with alternative ways for obtaining numerical results. In particular, as summarized in Table 4.1 of Wallace (2004), the simulation was compared with other methods for treating several different special cases. Among these were exact numerical results for the $M/M/C/K$ model and the bilingual call center analyzed by Stanford and Grassman (1993, 1998). For more complicated models, a comparison was made with Ridley's (2003) simulator for call centers with time-dependent arrival rates, specialized to the case of constant arrival rates.

We show the simulation results in Figure 1. (More detailed descriptions of system performance in the 18 cases are shown in tables in the Internet Supplement. Those tables show that the variability of the estimates is not great; e.g., the 95% confidence intervals for per-class probabilities are about 1%.) Figure 1 shows 9 individual graphs in a 3×3 arrangement. The columns correspond to the normal, light and heavy load cases, respectively. The rows show estimates of (i) the steady-state blocking probability, (ii) the conditional expected steady-state delay (before beginning service), given that the call enters (is not blocked), and (iii) the conditional steady-state probability that the delay exceeds 0.5 minutes, given that the call enters. Both the blocking probability and the delay probability are in percentages. For each of the offered-load scenarios, we make 6 different simulation runs, one for each different number of skills. The horizonal axis for each graph specifies the number of skills that the agents have. The full model requires specifying the agent-skill matrix, which we have done above. The six agent-skill matrices themselves are displayed in the Internet Supplement, Wallace and Whitt (2004).

Figure 1 dramatically shows the existence of resource pooling. With only one exception, there are significant improvements in performance when agents are given at least two skills. That one exception is the conditional waiting-time tail probability under heavy loading. However, in all cases we see that most of the benefit is achieved by adding the second skill. Only modest further improvements are achieved when additional skills are provided. Indeed, Figure 1 shows that near-full resource pooling is achieved by giving agents only two skills.

In order to put the simulation results into perspective, it is useful to analyze related cases for the $M/M/C/K$ model analytically. We do so in Table 1. First, when $m = 6$, the model is approximately equivalent to the $M/M/90/30$ model. As noted above, when $m = 1$, the model does not reduce to six separate $M/M/15/5$ models, because the six call types actually share the common waiting room of size 30. However, it is clear that the $M/M/15/5$ model is a worst-case bound for the blocking probabilities. On the other hand, the $M/M/15/30$ model

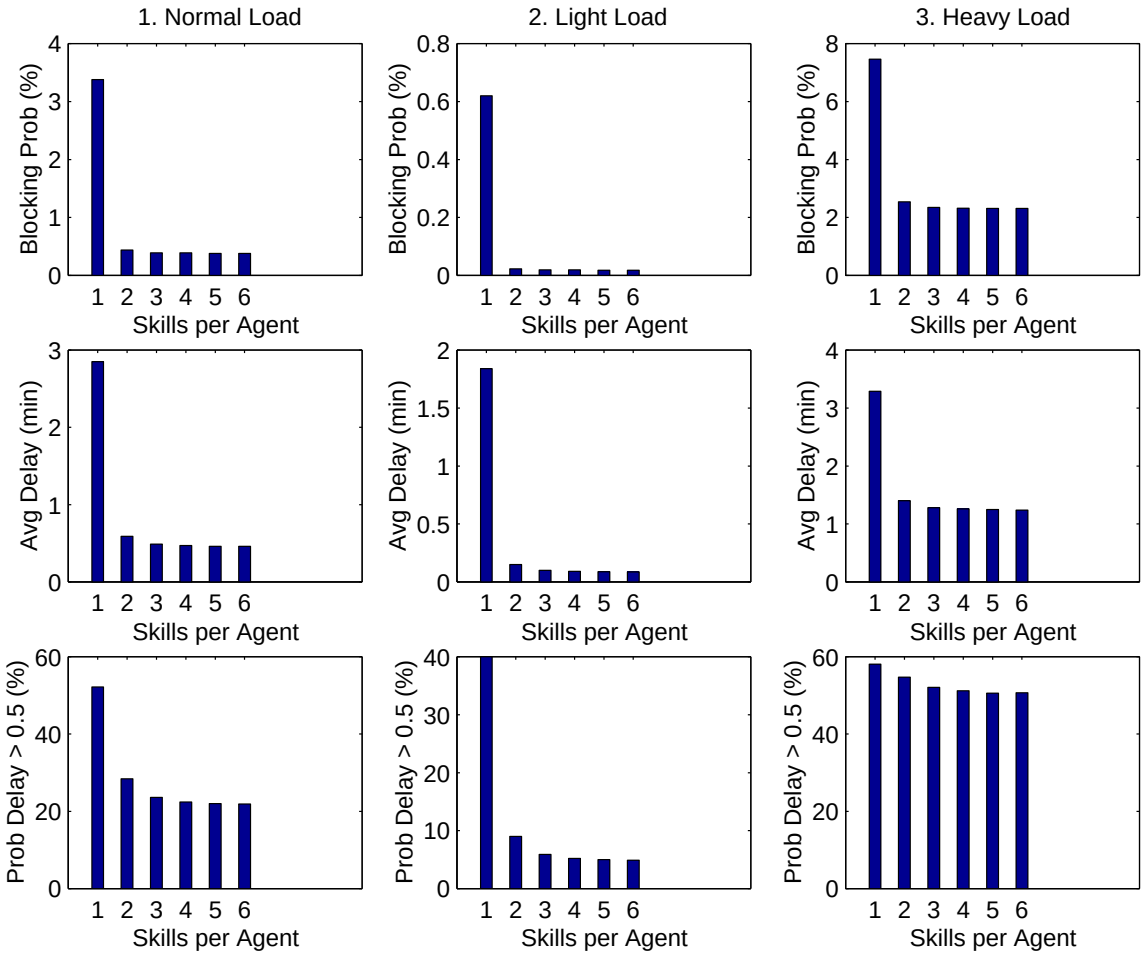

Figure 1: Typical performance measures as a function of the number of skills per agent and the offered load for an $M_6/M/90/30$ SBR call center with $1/\mu_1 = \cdots = 1/\mu_6 = 10$ minutes. The specific performance measures are the blocking probability, the mean conditional delay given entry, and the conditional probability that the delay is greater than 0.5, given entry.

is a best-case bound for the blocking probabilities. To put the simulation results for $m = 1$ in perspective, we thus calculate performance measures for the $M/M/15/5$ and $M/M/15/30$ models, with the same arrival and service rates.

From Table 1, we see that indeed the performance in the SBR model with 2 skills is close to both the performance in the SBR model with 6 skills and the performance in the $M/M/90/30$ model. On the other hand, the performance in these cases is not close to the performance in the other three cases, and the performance in those three cases varies widely.

5. The SBR Provisioning Algorithm

In this section we apply the resource-pooling property to develop an algorithm to generate requirements for staff and trunk lines in our call-center model with skill-based routing. That means that we aim to determine the parameters C and K in the $M_n/M/C/K/SBR$ model and the agent-skill matrix A, given the other model parameters and various constraints on A and on performance. Our main idea is to assume that there will be sufficient cross training so that resource pooling holds, but we use simulation to verify that the performance is indeed satisfactory, and to make appropriate adjustments if it is not.

The algorithm is carried out in **three steps**: (i) Exploit the $M/M/C/K$ model to find an *initial candidate solution*, (ii) add servers one at a time as necessary (determined by performing simulation experiments) to find an *initial feasible solution* (satisfying all the constraints) and (iii) perform a local search using additional simulation experiments to find a *better feasible solution*.

The local search is much in the spirit of Choudhury et al. (1995), but that algorithm exploited numerical transform inversion instead of simulation. The algorithm here is somewhat more complicated than it otherwise would be because we consider *both* the available waiting space K and the number of servers, C. Modifying the algorithm for the case $K = \infty$ is not difficult.

The Optimization Problem. Our primary goal is to minimize C, the total staff required. A secondary goal, for given C, is to minimize $C + K$, the required number of trunk lines. We seek minimum values of C and K subject to the condition that constraints on performance are satisfied. We consider the two sets of per-class performance constraints in (2.1) and (2.2). We also aim to determine an appropriate agent-skill matrix A, subject to constraints on it and subject to constraints on performance. *The specific problem we consider here allows an*

		Performance Measure			
Loading	Model	Blocking	$E[W]$	$P(W \leq 0.5)$	Utiliz.
light	$M/M/90/30$	0.00017	0.08	0.942	0.860
	6 skills	0.00018	0.09	0.951	0.860
	2 skills	0.00023	0.15	0.910	0.860
	1 skill	0.00620	1.84	0.600	0.854
	$M/M/15/5$	0.0388	0.59	0.737	0.854
	$M/M/15/30$	0.00007	2.15	0.575	0.827
normal	$M/M/90/30$	0.0036	0.45	0.733	0.930
	6 skills	0.0038	0.46	0.781	0.929
	2 skills	0.0044	0.59	0.716	0.928
	1 skill	0.0336	2.85	0.478	0.897
	$M/M/15/5$	0.0650	0.82	0.641	0.927
	$M/M/15/30$	0.0066	4.94	0.344	0.872
heavy	$M/M/90/30$	0.024	1.24	0.387	0.977
	6 skills	0.023	1.24	0.493	0.976
	2 skills	0.025	1.40	0.453	0.974
	1 skill	0.075	3.29	0.419	0.918
	$M/M/15/5$	0.095	1.05	0.555	0.972
	$M/M/15/30$	0.028	8.97	0.153	0.905

Table 1: A comparison of simulation estimates of performance measures for the SBR model with 6 skills, 2 skills and 1 skill for three different loadings: light, normal and heavy. Also included are the corresponding exact numerical results for the $M/M/90/30$, $M/M/15/5$ and $M/M/15/30$ models. The waiting time is conditional upon entry.

arbitrary agent-skill matrix A subject to the constraint that each agent have precisely two skills. We intend that to be illustrative of what can be done by our general approach, exploiting the resource pooling resulting from limited cross training, with appropriate sharing of skills.

Finding an Initial Candidate Pair (C,K). We initially act as if all agents have all skills. Thus we choose initial values of C and K to meet the performance requirements in a standard $M/M/C/K$ model, using the total aggregate arrival rate $\lambda \equiv \lambda_1 + \cdots + \lambda_n$. We also need to choose single performance constraints replacing the (in general) class-dependent performance constraints in (2.1) and (2.2). To be specific (and optimistic), choose the loosest among these constraints, or any one if the choice is not obvious. (This step should not be critical.)

Ways to find the optimal values of C and K for the $M/M/C/K$ model, along with supporting theory, have been developed by Massey and Wallace (2005). We apply that approach. We remark that the treatment of K is somewhat subtle. If we increase K to reduce blocking and improve performance, at the same time we will increase delays, and thus make performance worse from another perspective. Thus, when we change C by increasing or decreasing it by 1, it is advantageous to simultaneously adjust K so as to keep $C + K$ fixed.

Determining the Initial Work Group Sizes. Having used the $M/M/C/K$ model to find an initial pair (C, K), we now must specify the skills and priority levels for the C agents. We first determine the initial work group sizes; i.e., we first determine C_k, the number of agents in work group k, for each k, $1 \leq k \leq n$.

Motivated by heavy-traffic stochastic-process limits for many-server queues, e.g., Halfin and Whitt (1981), Puhalskii and Reiman (2000) and Garnett et al. (2002), we use a *square-root approximation* to allocate agents to the n work groups. In particular, first ignoring integrality constraints, we first generate real numbers R_k as initial estimates for C_k. We let

$$R_k = \alpha_k + x\sqrt{\alpha_k}, \quad 1 \leq k \leq n \ , \tag{5.1}$$

where $\alpha_k \equiv \lambda_k/\mu_k$ is the *offered load* of call type k and x is a positive constant. It is easy to see that x must be

$$x = \frac{(C - \alpha)}{\sum_{i=1}^{n} \sqrt{\alpha_i}} \ . \tag{5.2}$$

It is also easy to see that $R_k > 0$ and $R_1 + \cdots + R_n = C$ provided that $C > \alpha$, which will always be the case if the blocking probability is sufficiently small. As shown in Theorem 8.2.1 of Wallace (2004), the square-root method allocates relatively more capacity to the work groups

with the smaller offered loads than would an allocation that is directly proportional to the loads.

We next round the work group sizes R_k specified in (5.1) to obtain the desired integer values C_k, keeping the property $C_1 + \cdots + C_n = C$. When there are several work groups, none of which are large, this rounding can cause difficulties, but these difficulties will be addressed in the later phases of the algorithm, so we do not consider the rounding problem very important.

Here is One way to do the rounding: First round down, letting $C_k^* = \lfloor R_k \rfloor$, $1 \leq k \leq n$, where $\lfloor y \rfloor$ is the greater integer less than or equal to y. We start by assigning C_k^* agents to work group k. The total number of agents assigned so far is $C^* \equiv C_1^* + \cdots + C_n^*$. We then need to assign the remaining $C - C^*$ agents to work groups. (Note that necessarily $0 \leq C - C^* \leq n - 1$.) Next assign the $C - C^*$ remaining agents one to each group in order of the differences $R_k - \lfloor R_k \rfloor$ (higher numbers first).

Determining the Initial Agent-Skill Matrix. Given the work group sizes, we next need to construct an associated initial agent skill matrix A. We focus on the case in which all agents are given a secondary skill. When each agent is given a secondary skill, we first need to specify the secondary skill to go with the given primary skill. Here we assume all possibilities are open to us. To achieve that goal, we we use a *fair assignment rule*, aiming to "optimally" balance.

Let $C_{i,k}$ denote the number of agents in work group i (having primary skill i) that will be assigned secondary skill k. Let $R_{i,k}$ be an initial estimate of $C_{i,k}$, which may not be integer. For any i and k with $k \neq i$, we let

$$R_{i,k} = \frac{C_i C_k}{C - C_i} \ . \tag{5.3}$$

Note that the sum of $R_{i,k}$ over k is C_i. Also note that this rule lets the number of agents in work group i having secondary skill k be directly proportional to both C_i, the size of work group i, and C_k, the size of work group k. Finally, we round, as described above, to obtain the desired numbers $C_{i,k}$.

When each agent has at most two skills, the procedure above fully specifies the agent-skill matrix (except for irrelevant permutations of the rows). However, more is required when agents have more than two skills. In that case, we do not try to be more careful; we use the scheme described in Section 4. Hence at this point we have determined an initial candidate solution, but it may not be feasible, because some of the performance constraints may be violated.

Finding an Initial Feasible Solution. Since we have based our initial staffing on the resource-pooling property, we anticipate that the initial staffing requirement C is optimistic (a lower bound). But to find out, we simulate the call-center model, which has been fully specified above, and examine the performance. If we find that all the performance constraints are satisfied, then we have found an initial feasible solution, and we move on to the next step. However, we may well find that some of the performance constraints are violated. In that case we make adjustments until we obtain an initial feasible solution. We keep increasing C and correspondingly adding rows to the matrix A until the constraints are met. But we also may need to adjust K, so the overall procedure is more complicated. We now explain how to make the adjustments. There are two steps in this phase of the algorithm:

Step 1: Increasing C. We start by doing the performance evaluation. Then we do the delay test: If the speed-to-answer service-level target is not met for one or more classes, we increment C by one and decrement K by 1, thus keeping $C + K$ fixed, and repeat the experiment. (See Massey and Wallace (2005) for motivation for this treatment of K.) Now we need to add a row to A.

We use a *performance-based method*; it adds a row with primary and secondary skills corresponding to the call types with the worst performance. To specify that, let

$$D_k \equiv \delta_k - P(W_k \le \tau_k | Q_k < C + K) \ . \tag{5.4}$$

Noting that positive is bad in (5.4), we let

$$\begin{aligned} k_1^* &= argmax\{D_k : 1 \le k \le n\} \\ k_2^* &= argmax\{D_k : 1 \le k \le n, \quad k \ne k_1^*\} \ , \end{aligned} \tag{5.5}$$

where $argmax$ identifies the index yielding the maximum. We thus add the row with primary skill k_1^* and secondary skill k_2^*. For example, when there are 6 call types, we add the row $(k_1^*, k_2^*, 0, 0, 0, 0)$. We repeat this step until all the speed-to-answer service-level targets are met. Because of the resource pooling, we anticipate that we will not need to increment C many times.

Step 2: Increasing K. Once we have found a staffing level C meeting the speed-to-answer service-level target, we investigate the blocking-probability target. If the blocking-probability target is met, then we have an initial feasible solution, and we can go on to the refinement phase (finding a better feasible solution). If the blocking-probability target is not met, then we increment K by one, and repeat Step 1. That is, we apply simulation to do another

performance evaluation. We see whether or not the speed-to-answer service-level targets are still met. They might not be, because increasing K can lower the blocking probabilities, but at the same time increase delays. So we repeat Step 1, increasing C if necessary. We go back and forth between Steps 1 and 2 until we see that both sets of performance constraints are met. When they are, we have obtained an initial feasible solution. (It is clear that this phase of the algorithm will terminate, because the parameters C and K are successively increased at each step.) We now move on to search for better feasible solutions.

Finding Better Feasible Solutions. In this final phase of the algorithm we perform a local search in order to find better feasible solutions, by which we mean solutions with smaller values of C, and for that C, the smallest possible value of K. We keep trying to lower C until we exhaust all possibilities (explained below). We have an important frame of reference, we already know the optimal pair (C, K) for the $M/M/C/K$ model. If we get close to this pair, we know that we must be nearly optimal.

We perform two steps: (i) We try to *remove* a row from A and still obtain a feasible solution, and (ii) given an infeasible A, we try to *change* a row, in order to make it feasible. If we are successful in the removal step, then we repeat it. If we are unsuccessful in the removal step, then we perform a change step. If we are unsuccessful in a change step, then we try another change step until we exhaust the allowed possibilities (specified below). If we are successful in one of these change steps, then we go back to the removal step. It thus suffices to specify the removal step and the change step, including the termination condition for the change step when we are unsuccessful.

Removal Step. In this step we decrement C and increment K, keeping $C + K$ fixed, and remove a row from A. We now specify how to determine the row to remove. We remove the row having primary and secondary skills with the best current performance. Paralleling (5.5), we specify that by letting

$$\begin{aligned} k_1^{**} &= argmin\{D_k : 1 \leq k \leq n, \quad A_{i,1} = k \quad \text{for some} \quad i\} \\ i^* &= argmin\{D_{A_{i,2}} : 1 \leq i \leq C, \quad A_{i,1} = k_1^{**}\} \ , \end{aligned} \tag{5.6}$$

for D_k in (5.4). We thus remove row i^*.

Having removed row i^*, we evaluate the performance of the new agent-skill matrix without that row. If all constraints are satisfied, then we make the current solution the best feasible solution so far, and repeat the removal step. If delay constraints are violated, we check to see

if they can be satisfied, with the blocking-probability constraints satisfied, by decreasing K. If that cannot be done, we conclude that removing row i^* was unsuccessful. We then move on to the change step, trying to make this latest infeasible solution feasible.

Change Step. In this step we initially leave C and K unchanged, and change an existing row of the agent-skill matrix A. We change a row by identifying *both* a row to remove, following (5.6), and a row to add, following (5.5). We thus want to remove primary skill $A_{i^*,1}$ and secondary skill $A_{i^*,2}$, while we want to add primary skill k_1^* and secondary skill k_2^*. The natural direct action is the double change: $(A_{i^*,1}, A_{i^*,2}) \to (k_1^*, k_2^*)$. If all performance constraints are satisfied, then we make the current feasible solution the best feasible solution so far.

If the delay constraints are satisfied, but the blocking constraints are not satisfied, then we try to get all constraints satisfied by successively increasing K. If that succeeds, then we make the final solution the best feasible solution so far. However, if that does not succeed, we terminate with the smallest value of K for which the delay constraints were not met. We then continue on to another change step.

Because there is no guarantee that the algorithm will terminate, we specify a maximum allowable number of these change steps, e.g., 20, trying to find a feasible solution. If no feasible solution is found within that number of steps, then the search is terminated. If the procedure returns to a previous case, then the search can be terminated as well.

If a feasible solution is found, then it becomes the best feasible solution so far, and we go back to the removal step. If no feasible solution is found, we stop.

Good feasible solutions can be recognized in two ways: (i) by the fact that all the performance constraints tend to be satisfied with relatively little slack and (ii) the (C, K) pair differs little from the optimal solution for the corresponding $M/M/C/K$ model.

The algorithm has now been fully specified. Many variations in the algorithm can be considered. In our implementations of the change step, we have confined changes to the double changes specified above, but we could also consider the following less bold changes, e.g., $(A_{i^*,1}, A_{i^*,2}) \to (k_1^*, A_{i^*,1})$, $(A_{i^*,1}, A_{i^*,2}) \to (k_1^*, A_{i^*,2})$, and $(A_{i^*,1}, A_{i^*,2}) \to (A_{i^*,1}, k_1^*)$. That has not proved necessary in our experiments.

6. A Balanced Example

In this section we show how the provisioning algorithm in Section 5 works in a balanced example, closely related to the example used for the resource-pooling experiment in Section

	M/M/C/K and $\lambda = 8.25$			
Perf. Measure	$C = 90, K = 21$	$C = 90, K = 20$	$C = 90, K = 19$	$C = 89, K = 21$
1. Blocking (%)	0.45	0.49	0.53	0.60
2. Mean Delay (min)	0.248	0.238	0.227	0.303
4. $\mathcal{P}\,(\text{Delay} \leq 0.5\,\vert\text{entry})$ (%)	82.4	82.9	83.2	78.9
4. $\mathcal{P}\,(\text{Delay} \leq 1.0\,\vert\text{entry})$ (%)	89.6	90.0	90.5	87.0
6. Avg. Agent Util. (%)	91.25	91.22	91.18	91.12

Table 2: Performance Measures for the $M/M/C/K$ model for different C and K to put the SBR simulation results in perspective. The mean service time is $1/\mu = 10$ minutes.

4. Like Section 4, the balanced example has 6 call types. Again the mean service time is 10 minutes. Here the offered load for each call type is 13.75, so that the total offered load is 82.5.

We consider the two performance constraints in (2.1) and (2.2). Here we let $\tau_k = \tau = 0.5$ and $\delta_k = \delta = 0.80$ for all k, which corresponds to the requirement that 80% of the calls of each type be answered within 0.5 minute (30 seconds).

Under the assumption of Poisson arrivals (which we have assumed here, but which need not hold in practice), the random variables Q_k are distributed as Q, the steady-state total number in the system at an arbitrary time, by virtue of the *Poisson Arrivals See Time Averages (PASTA)* property; e.g., see Wolff (1989). Here we will let the blocking probability target be $\epsilon_k = \epsilon = 0.005$ for all k, which corresponds to 0.5% blocking. Typically, agents are much more expensive than trunk lines, so that the blocking probability target should be relatively small. The blocking-probability constraint is included so that there are not substantially more trunk lines than needed.

We start by applying the $M/M/C/K$ model. First, the asymptotic method for the $M/M/C/K$ model in Massey and Wallace (2005) yields an initial solution of $(C, K) = (90, 21)$. As shown in Table 2 exact analysis yields $(90, 20)$ instead, but the blocking probability is close to the boundary (0.5% blocking). Clearly, 89 agents are insufficient; the delay constraint and the blocking constraint are simultaneously violated.

In contrast, if all agents have only one skill, then we have 6 separate $M/M/C/K$ models each with offered load 13.75, for which the optimal solution is $(18, 10)$, yielding a total $(C, K) = (108, 60)$. Of course, since we have a common waiting room, we would not need $K = 60$. The main point is that we would need 18 more agents if each agent had only one skill. So, clearly, adding multiple skills has an enormous performance impact. Indeed, 20% more agents are required when agents have only one skill instead of all six skills. The question is whether we

can experience much less performance degradation when agents have only two skills.

We now turn to the simulation phase of the algorithm. We now apply our provisioning algorithm. For the initial value $C = 90$, the algorithm assigns primary and secondary skills in a balanced way: Each of the 6 work groups has 15 agents and, for each work group (primary skill), each of the 5 remaining skills are assigned to 3 agents in the work group. Thus each of the 30 pairs of distinct skills are assigned as primary and secondary skills to 3 agents. Given that obvious initial agent skill matrix, we apply simulation to evaluate its performance.

The simulation results for this initial case with $(C, K) = (90, 21)$ and that skill matrix is given in Table 3. We show the overall blocking probability, the overall mean delay, the mean delay for each call type, the overall speed-to-answer service-level $\mathcal{P}\left(\text{Delay} \leq 0.5 \,|\text{entry}\right)$, the per-call-type speed-to-answer service-level $\mathcal{P}\left(\text{Delay}_i \leq 0.5 \,|\text{entry}\right)$, the overall agent utilization percentage, the work-group utilization percentages and the percentage work-group utilizations devoted to call types with that skill as a primary skill. We do not display confidence intervals, but since the model is symmetric, we can see the statistical precision from the six per-class results.

From Table 3, we see that the blocking probability is 0.0054, which is above the target $\epsilon = 0.0050$, while some of the speed-to-answer service-level probabilities are also below the target 80%. However, the constraints are only violated by very small amounts. Indeed, those speed-to-answer discrepancies could conceivably be due to statistical error, but the blocking gap seems to be statistically (if not practically) significant. For all practical purposes, the initial candidate solution based on the $M/M/C/K$ model does the job for this balanced example.

However, we will proceed until all constraints are fully satisfied. Following the specified procedure, we increment C by 1 and decrement K by one, to produce the new candidate pair $(91, 20)$. It next remains to specify the work groups and the agent-skill matrix. For this symmetric example, we use the performance-based method and select the agents with the worst performance. We thus add one agent to work group 6 with secondary skill 5, i.e., we add the row $(6, 5, 0, 0, 0, 0)$ to the initial agent-skill matrix. Then we simulate this new case and obtain the results for iteration 2 in Table 3. Now we see that the performance constraints are all satisfied. We continue for two more iterations to see if we can reduce the number of waiting spaces. We arrive at $(91, 19)$ as the best solution in iteration 3, as confirmed by iteration 4. As we should expect, the performance is somewhat better for work groups 6 and 5 because they received the extra help. The main point, though, is that the provisioning solution when each agent has only two skills is nearly the same as for the single-class $M/M/C/K$ model. *There*

is only a difference of a single agent!

To further investigate what is going on, we also apply our staffing algorithm to the case in which all agents are universal agents, i.e., in which all agents have 6 skills. Interestingly, we find that the best feasible solution has a strictly smaller value of C than for the $M/M/C/K$ model. Our provisioning algorithm terminates with the pair $(C, K) = (89, 24)$. It turns out that the non-FCFS service discipline of the 6-skill policy requires one fewer agent (but with extra trunk lines). We performed additional simulation experiments to verify that this is a bonafide phenomenon of the routing policy. Even so, the staffing with two skills differs by only 2 agents from the staffing with six skills. Moreover, the application of the $M/M/C/K$ model to find an initial candidate solution gets very close to the final answer. Only one more iteration was required to reach the best feasible solution.

7. An Unbalanced Example

In this section we consider a more difficult unbalanced example. We leave the mean service times the same, but modify the arrival rates so that the offered loads become

$$\alpha_1 = \alpha_2 = 4.25, \quad \alpha_3 = 10.50, \quad \alpha_4 = 13.75, \quad \alpha_5 = 19.25 \quad \text{and} \quad \alpha_6 = 30.50 \tag{7.1}$$

Just as for the balanced example, the total offered load is 82.50, but now the six offered loads are unbalanced. We use the same (common) performance constraints as before.

Just as for the balanced example, the solution based on the $M/M/C/K$ model is $(C, K) = (90, 21)$. On the other hand, if each agent has only a single skill, then we find the six required (C, K) pairs associated with the offered loads in (7.1) are: $(7, 5)$, $(7, 5)$, $(14, 8)$, $(18, 9)$, $(24, 10)$, and $(36, 13)$, respectively, yielding a total requirement of $(106, 50)$. Interestingly, fewer agents are required in the single-skill case for the unbalanced model than for the balanced model (106 instead of 108), but there still is a dramatic increase in required resources.

To see what happens when each agent has two skills, we again turn to the simulation. However, in the unbalanced case the initial case is no longer symmetric, so from the outset we have rounding problems when we specify the work groups. The second step to find an initial feasible solution phase takes four iterations, proceeding from $C = 90$ to $C = 93$.

As indicated, the initial phase of the simulation algorithm takes us to the initial feasible solution $(93, 18)$ in four steps. We then proceed to obtain better feasible solutions. We reach $(91, 20)$, just as in the balanced example. That takes four iterations in the order: remove, change, remove, change. However, subsequent iterations (remove, change, ...) produce no

The SBR Provisioning Algorithm for the Balanced Example					
	No. of Iterations (C agents, K waiting spaces)				
Performance Measure	**1 (90, 21)**	**2 (91, 20)**	**3 (91, 19)**	**4 (91, 18)**	**6 skills (89, 24)**
1. Blocking (%)	0.54	0.43	0.49	0.54	0.47
2. Mean Delay (min)	0.36	0.30	0.29	0.27	0.34
3. Mean Delay 1	0.37	0.32	0.30	0.29	0.33
3. Mean Delay 2	0.36	0.32	0.30	0.28	0.31
3. Mean Delay 3	0.35	0.30	0.29	0.28	0.37
3. Mean Delay 4	0.36	0.30	0.29	0.28	0.33
3. Mean Delay 5	0.35	0.28	0.27	0.26	0.34
3. Mean Delay 6	0.37	0.28	0.27	0.26	0.34
4. $\mathcal{P}\left(\text{Delay} \le 0.5 \,\middle|\text{entry}\right)$ (%)	79.8	82.7	83.1	83.6	82.2
5. $\mathcal{P}\left(\text{Delay}_1 \le 0.5 \,\middle|\text{entry}\right)$	80.3	81.9	82.4	83.0	82.4
5. $\mathcal{P}\left(\text{Delay}_2 \le 0.5 \,\middle|\text{entry}\right)$	79.7	81.9	82.7	83.5	82.6
5. $\mathcal{P}\left(\text{Delay}_3 \le 0.5 \,\middle|\text{entry}\right)$	80.0	82.5	82.7	83.2	81.9
5. $\mathcal{P}\left(\text{Delay}_4 \le 0.5 \,\middle|\text{entry}\right)$	80.5	82.6	82.9	83.4	82.5
5. $\mathcal{P}\left(\text{Delay}_5 \le 0.5 \,\middle|\text{entry}\right)$	79.7	83.5	83.9	84.3	82.2
5. $\mathcal{P}\left(\text{Delay}_6 \le 0.5 \,\middle|\text{entry}\right)$	79.5	83.9	84.1	84.5	82.1
6. Avg Agent Util (%)	91.1	90.2	90.2	90.1	92.0
7. Work Group 1 Util	91.2	90.4	90.2	90.2	91.9
7. Work Group 2 Util	91.2	90.3	90.3	90.2	92.0
7. Work Group 3 Util	91.1	90.4	90.3	90.2	92.2
7. Work Group 4 Util	91.2	90.3	90.3	90.2	92.0
7. Work Group 5 Util	91.2	90.4	90.3	90.3	92.0
7. Work Group 6 Util	91.2	89.7	89.7	89.8	91.8
8. Work Group 1 Prim Util	68.3	69.1	68.8	68.5	61.0
8. Work Group 2 Prim Util	68.0	68.9	68.7	68.2	61.2
8. Work Group 3 Prim Util	67.8	68.7	68.9	68.4	62.7
8. Work Group 4 Prim Util	67.8	68.9	68.5	68.2	61.3
8. Work Group 5 Prim Util	67.9	68.3	67.7	68.3	61.4
8. Work Group 6 Prim Util	68.3	66.5	66.7	66.7	61.2

Table 3: Performance measures for the balanced-offered-load example in which the offered load are $\alpha_1 = \cdots = \alpha_6 = 13.75$, the mean service times are $1/\mu_1 = \cdots = 1/\mu_6 = 10.0$ min, the blocking-probability target is $\epsilon = 0.005$ and the speed-to-answer service-level target is $\mathcal{P}\left(\text{Delay} \le 0.5 \,\middle|\text{entry}\right) \ge 0.80$. The last column gives the six-skill best feasible solution for comparison.

improvement. Indeed, in the $(90, 21)$ case obtained in the next remove step, both delay and blocking constraints are violated, without any satisfied constraints having significant slack.

Just as in the balanced example, we also consider the case in which all agents have all six skills. Once again, when all agents have all six skills, the best feasible solution has $C = 89$. For this example, $K = 24$. Thus we have admirably achieved our goal. The final feasible solution is within a single agent of what can be achieved for the $M/M/C/K$ model and within two agents of what can be achieved when all agents have all skills.

8. Extensions

In this final section we discuss some extensions.

Class-Dependent Performance Constraints. All the experiments described above have performance constraints the same for all customer classes. We also did experiments with class-dependent performance constraints. In particular, we introduced three *grades of service*: platinum, gold and silver. Among our six classes, we let two classes be given each grade of service. We specified the meaning by letting the service-level-constraint parameter pairs (τ_k, δ_k) in (2.1) be, respectively, $(90, 20)$, $(80, 20)$ and $(80, 30)$. We left the rest of the balanced and unbalanced models unchanged, including the blocking-constraint parameters. We used the coarsest pair $(80, 30)$ in the $M/M/C/K$ analysis, which was just as before. Thus the initial staffing for the $M/M/C/K$ model is unchanged: $(90, 21)$.

Just as before, the algorithm converged after several iterations. In particular, for the unbalanced example, four iterations were required to find the initial feasible solution with $(C = 94, K = 17)$. The best feasible solution was obtained at iteration 15, yielding $(C = 93, K = 15)$. The results for 14 of the 16 iterations are shown in Tables 4 and 5. The candidate solution with $(C = 92, K = 19)$ obtained at iteration 11 has all but one delay constraint violated, with that one quite close, so that our final candidate $(C = 93, K = 15)$ is likely to be the optimal solution. The higher staff requirement is as expected, because some of the performance constraints are more stringent.

Other Performance Constraints. Our approach applies equally well to other performance constraints besides the ones in (2.1) and (2.2), class-dependent or not. For example, we could have constraints on the expected per-class conditional steady-state waiting time, $E[W_k|Q_k <$

SBR Heuristic Resource Provisioning Algorithm							
	No. of Iterations (No. of Agents, Extra Waiting Space)						
Performance Measure	**1 (90,21)**	**2 (91,20)**	**3 (92,19)**	**4 (93,18)**	**5 (94,17)**	**6 (93,18)**	**7 (93,18)**
1. Blocking (%)	0.53	0.42	0.36	0.31	0.26	0.30	0.31
2. Mean Delay (min)	0.35	0.29	0.23	0.19	0.15	0.19	0.19
3. Mean Delay 1	0.80	0.67	0.58	0.39	0.35	0.39	0.41
3. Mean Delay 2	0.84	0.57	0.48	0.40	0.30	0.35	0.35
3. Mean Delay 3	0.37	0.27	0.21	0.15	0.12	0.15	0.13
3. Mean Delay 4	0.38	0.32	0.27	0.23	0.19	0.24	0.26
3. Mean Delay 5	0.27	0.24	0.18	0.16	0.11	0.14	0.14
3. Mean Delay 6	0.24	0.21	0.16	0.14	0.12	0.15	0.16
4. $\mathcal{P}\,(\text{Delay} \leq 0.5\,\vert\text{entry}\,)$ (%)	81.1	83.7	86.5	88.6	90.6	88.5	88.4
5. $\mathcal{P}\,(\text{Delay}_1 \leq 0.5\,\vert\text{entry}\,)$	68.2	72.0	74.9	81.2	83.3	81.7	81.3
5. $\mathcal{P}\,(\text{Delay}_2 \leq 1/3\,\vert\text{entry}\,)$	65.0	72.3	75.1	78.1	82.2	80.1	80.6
5. $\mathcal{P}\,(\text{Delay}_3 \leq 1/3\,\vert\text{entry}\,)$	76.9	81.4	85.1	88.2	90.2	88.2	89.3
5. $\mathcal{P}\,(\text{Delay}_4 \leq 1/3\,\vert\text{entry}\,)$	76.0	78.7	81.3	83.8	86.0	82.9	81.8
5. $\mathcal{P}\,(\text{Delay}_5 \leq 1/3\,\vert\text{entry}\,)$	79.9	82.0	85.7	87.5	90.1	88.2	88.7
5. $\mathcal{P}\,(\text{Delay}_6 \leq 0.5\,\vert\text{entry}\,)$	84.4	86.3	88.9	90.4	92.9	89.8	89.3
6. Avg Agent Util (%)	91.2	90.2	89.3	88.4	87.5	88.4	88.4
7. Work Group 1 Util	86.6	85.4	84.1	81.3	79.9	81.2	81.1
7. Work Group 2 Util	86.5	84.2	82.7	81.2	79.9	81.1	80.8
7. Work Group 3 Util	89.6	88.5	87.0	86.1	85.2	86.1	85.4
7. Work Group 4 Util	90.7	90.0	89.1	88.6	87.7	88.6	88.7
7. Work Group 5 Util	91.6	90.9	90.2	89.4	88.2	89.0	89.2
7. Work Group 6 Util	93.1	92.5	91.8	91.3	90.7	91.7	92.1
8. Work Group 1 Prim Util	53.8	54.7	55.5	51.7	52.5	51.4	51.6
8. Work Group 2 Prim Util	53.6	50.3	51.4	51.3	51.9	50.9	50.7
8. Work Group 3 Prim Util	63.4	62.8	61.5	61.1	62.2	61.3	59.0
8. Work Group 4 Prim Util	68.9	69.2	70.2	71.0	71.3	71.2	71.3
8. Work Group 5 Prim Util	71.6	72.5	72.7	73.0	72.0	71.4	70.8
8. Work Group 6 Prim Util	77.9	78.5	78.7	79.5	79.8	80.8	81.8

Table 4: Performance measures in the first 7 iterations of the SBR heuristic algorithm in the case of different service-level targets and unbalanced offered loads: $\rho_1 = \rho_2 = 4.25$, $\rho_3 = 10.50$, $\rho_4 = 13.75$, $\rho_5 = 19.25$, and $\rho_6 = 30.50$. The mean service times are $1/\mu_1 = \cdots = 1/\mu_6 = 10.0$ min. and the target blocking is $\epsilon = 0.005$. The service-level targets for call types 1 through 6 are (80, 30), (80, 20), (90, 20), (80, 20), (90,20), and (80, 30), respectively, where (80, 30) means $\mathcal{P}\,(\text{Delay} \leq 0.5\,\vert\text{entry}\,) \geq 0.80$.

SBR Heuristic Resource Provisioning Algorithm							
	No. of Iterations (No. of Agents, Extra Waiting Space)						
Performance Measure	**8 (93,18)**	**9 (93,18)**	**10 (93,18)**	**11 (93,18)**	**12 (92,19)**	**15 (93,15)**	**16 (93,14)**
1. Blocking (%)	0.31	0.32	0.32	0.32	0.38	0.44	0.503
2. Mean Delay (min)	0.20	0.20	0.20	0.21	0.26	0.19	0.18
3. Mean Delay 1	0.40	0.45	0.34	0.38	0.44	0.34	0.34
3. Mean Delay 2	0.34	0.35	0.33	0.36	0.40	0.31	0.32
3. Mean Delay 3	0.13	0.12	0.12	0.11	0.14	0.10	0.09
3. Mean Delay 4	0.28	0.28	0.29	0.25	0.31	0.24	0.23
3. Mean Delay 5	0.12	0.11	0.10	0.11	0.12	0.09	0.09
3. Mean Delay 6	0.18	0.19	0.22	0.24	0.32	0.22	0.20
4. $\mathcal{P}\,(\text{Delay} \le 0.5\,\vert\text{entry}\,)$ (%)	88.2	887.9	87.6	87.5	84.9	88.2	88.6
5. $\mathcal{P}\,(\text{Delay}_1 \le 0.5\,\vert\text{entry}\,)$	81.4	79.9	82.9	81.8	79.5	83.0	82.6
5. $\mathcal{P}\,(\text{Delay}_2 \le 1/3\,\vert\text{entry}\,)$	80.5	80.2	80.5	80.0	78.3	81.2	81.4
5. $\mathcal{P}\,(\text{Delay}_3 \le 1/3\,\vert\text{entry}\,)$	89.8	90.2	90.1	90.7	88.9	91.6	91.7
5. $\mathcal{P}\,(\text{Delay}_4 \le 1/3\,\vert\text{entry}\,)$	81.0	80.9	80.0	81.9	78.6	82.6	82.8
5. $\mathcal{P}\,(\text{Delay}_5 \le 1/3\,\vert\text{entry}\,)$	89.4	90.1	90.6	90.6	89.0	91.4	91.8
5. $\mathcal{P}\,(\text{Delay}_6 \le 0.5\,\vert\text{entry}\,)$	88.3	87.3	86.2	85.3	81.3	85.6	86.4
6. Avg Agent Util (%)	88.4	88.4	88.4	88.4	89.3	88.3	88.2
7. Work Group 1 Util	80.8	81.3	79.8	79.9	81.2	79.6	79.8
7. Work Group 2 Util	80.8	81.0	80.9	80.9	82.4	80.8	80.6
7. Work Group 3 Util	85.4	85.4	85.7	85.9	87.1	85.7	85.6
7. Work Group 4 Util	88.9	89.0	89.3	88.5	89.5	88.6	88.5
7. Work Group 5 Util	88.6	88.2	88.3	88.5	89.3	88.3	88.2
7. Work Group 6 Util	92.6	92.8	93.3	93.6	94.3	93.5	93.5
8. Work Group 1 Prim Util	51.4	52.5	48.2	48.9	48.6	48.6	49.3
8. Work Group 2 Prim Util	50.8	51.5	50.8	51.1	51.0	51.0	50.8
8. Work Group 3 Prim Util	58.7	58.0	58.0	57.7	56.8	57.8	57.6
8. Work Group 4 Prim Util	71.4	71.3	71.1	69.1	68.6	69.3	68.9
8. Work Group 5 Prim Util	69.1	67.6	66.9	67.1	66.0	66.7	66.6
8. Work Group 6 Prim Util	83.1	84.1	85.4	86.4	87.4	86.4	86.1

Table 5: Performance measures in 7 later iterations of the SBR heuristic algorithm in the case of different service-level targets and unbalanced offered loads: $\rho_1 = \rho_2 = 4.25$, $\rho_3 = 10.50$, $\rho_4 = 13.75$, $\rho_5 = 19.25$, and $\rho_6 = 30.50$. The mean service times are $1/\mu_1 = \cdots = 1/\mu_6 = 10.0$ min. and the target blocking is $\epsilon = 0.005$. The service-level targets for call types 1 through 6 are (80, 30), (80, 20), (90, 20), (80, 20), (90,20), and (80, 30), respectively, where (80, 30) means $\mathcal{P}\,(\text{Delay} \le 0.5\,\vert\text{entry}\,) \ge 0.80$.

$C+K]$ (the average speed of answer, ASA, for answered calls), or we could have two (or more) different per-class speed-to-answer service-level constraints like the one in (2.1), one focusing on relatively small targets, such as 30 seconds, and another focusing on larger targets, such as 3 minutes. We might want to ensure that 80% of all calls are answered within 30 seconds and that 99% of all calls are answered within 3 minutes. The second speed-to-answer service-level constraint ensures that callers failing to meet the first target are not subsequently given poor service.

Class-Dependent Mean Service Times. A critical assumption so far in this paper is that all service times have a common mean. It is easy to extend the algorithm to other cases, but there is good reason to question whether it will still perform well, especially when the mean service times are allowed to depend on both the customer type and the agent; e.g., see Garnett and Mandelbaum (2000).

However, we note that the algorithm extends easily to class-dependent mean service times (independent of the agent). To find the initial candidate solution by applying the $M/M/C/K$ model, we then need to define an overall mean service time. That is easily done by letting the expected service time be a convex combination of the individual mean service times, namely,

$$\frac{1}{\mu} \equiv \frac{1}{\lambda} \sum_{i=1}^{n} \frac{\lambda_i}{\mu_i} \ . \tag{8.1}$$

The overall service-time is now hyperexponential (a mixture of exponentials) instead of exponential, so the $M/M/C/K/$ model is even more optimistic, but it can nevertheless be used, because it is only being used to generate an initial candidate solution.

We investigated several cases of unequal mean service times. We considered the same balanced and unbalanced examples, where two of the classes had mean service times $10 - x$, two of the classes had mean service times 10, and two of the classes had mean service times $10 + x$, for $x = 2$ and $x = 7$. Surprisingly, perhaps, the algorithm was also effective in these alternative scenarios, yielding nearly the same staffing (within a single agent). To illustrate, we present the detailed performance measures for the balanced example with $x = 2$ in Table 6; the mean service times were thus 8, 10 and 12. The optimal solution was $(91, 19)$, just as in the balanced example in Table 3.

However, it remains to carefully evaluate how the algorithm performs under such generalizations. Bad behavior is clearly possible when the mean service times vary greatly and depend upon both the call type and the agent.

SBR Heuristic Resource Provisioning Algorithm							
	Number of Iterations (No. of Agents, Extra Waiting Space)						
Performance Measure	**1 (90,21)**	**2 (91,20)**	**3 (92,19)**	**4 (91,20)**	**5 (90,21)**	**6 (91,19)**	**7 (91,18)**
1. Blocking (%)	0.56	0.44	0.38	0.45	0.56	0.49	0.55
2. Mean Delay (min)	0.37	0.31	0.25	0.30	0.36	0.29	0.28
3. Mean Delay 1	0.50	0.36	0.30	0.33	0.39	0.33	0.32
3. Mean Delay 2	0.49	0.40	0.28	0.33	0.38	0.32	0.32
3. Mean Delay 3	0.35	0.31	0.26	0.32	0.38	0.30	0.30
3. Mean Delay 4	0.36	0.32	0.27	0.32	0.37	0.30	0.29
3. Mean Delay 5	0.27	0.23	0.19	0.26	0.35	0.25	0.24
3. Mean Delay 6	0.27	0.23	0.20	0.25	0.32	0.24	0.23
4. $\mathcal{P}\,(\text{Delay} \leq 0.5\,\vert\text{entry}\,)$ (%)	79.4	82.5	85.2	82.6	79.8	83.0	83.3
5. $\mathcal{P}\,(\text{Delay}_1 \leq 0.5\,\vert\text{entry}\,)$	74.5	79.8	82.9	81.2	78.4	81.2	81.5
5. $\mathcal{P}\,(\text{Delay}_2 \leq 0.5\,\vert\text{entry}\,)$	74.7	78.6	83.5	80.9	78.5	81.4	81.4
5. $\mathcal{P}\,(\text{Delay}_3 \leq 0.5\,\vert\text{entry}\,)$	79.9	82.2	84.9	81.9	79.4	82.4	82.7
5. $\mathcal{P}\,(\text{Delay}_4 \leq 0.5\,\vert\text{entry}\,)$	80.2	82.4	84.5	82.2	79.9	82.7	82.8
5. $\mathcal{P}\,(\text{Delay}_5 \leq 0.5\,\vert\text{entry}\,)$	83.8	86.1	88.3	84.8	80.9	85.0	85.6
5. $\mathcal{P}\,(\text{Delay}_6 \leq 0.5\,\vert\text{entry}\,)$	83.6	86.1	87.4	84.7	82.0	85.6	85.8
6. Avg Agent Util (%)	91.1	90.2	89.3	90.2	91.1	90.2	90.1
7. Work Group 1 Util	90.6	89.1	88.2	89.0	90.1	89.1	89.0
7. Work Group 2 Util	90.7	89.5	88.0	89.1	90.1	89.0	88.9
7. Work Group 3 Util	91.1	90.4	89.5	90.3	91.0	90.2	90.3
7. Work Group 4 Util	91.0	90.3	89.5	90.4	91.2	90.4	90.2
7. Work Group 5 Util	91.4	90.8	89.9	91.2	92.4	91.2	91.1
7. Work Group 6 Util	91.5	90.6	89.9	90.7	91.6	90.7	90.6
8. Work Group 1 Prim Util	67.5	64.7	65.5	64.2	53.8	64.6	64.3
8. Work Group 2 Prim Util	67.6	67.0	64.9	64.3	51.6	64.3	64.3
8. Work Group 3 Prim Util	68.3	68.9	69.5	69.0	67.6	68.4	68.8
8. Work Group 4 Prim Util	69.2	69.0	69.5	68.9	67.6	68.7	68.5
8. Work Group 5 Prim Util	69.1	69.7	70.1	71.5	72.0	71.7	71.5
8. Work Group 6 Prim Util	69.0	69.4	70.2	70.3	80.5	69.8	69.5

Table 6: Performance measures using the SBR heuristic algorithm for the case of different mean service times, with balanced call volumes: $\lambda_1 = \cdots = \lambda_6 = 1.375$ calls/min, mean service times $1/\mu_1 = 1/\mu_2 = 8.0$ min, $1/\mu_3 = 1/\mu_4 = 10.0$ min, $1/\mu_5 = 1/\mu_6 = 12.0$ min, target blocking $\epsilon = 0.005$ and target delay $\mathcal{P}\,(\text{Delay} \leq 0.5\,\vert\text{entry}\,) = 1$ - $\mathcal{P}\,(\text{Delay} > 0.5\,\vert\text{entry}\,) = 0.80$.

Non-Exponential Distributions and Abandonments. We anticipate that the algorithm will be equally effective for generalizations of the present model in which customers may abandon before starting service, and where both the service times and patience times (times to abandon before starting service) have non-exponential distributions. These are important model generalizations, because they occur in real call centers and because they can make a big difference in performance. Statistical analysis of call-center data has shown that the service-time and patience distributions are often not nearly exponential; see Brown et al. (2002).

Simulations can easily incorporate these important features. It remains to verify that the performance of the algorithm is indeed good though. However, as indicated above, in order to achieve good performance, it may be important to require that the service-time distribution and patience distributions are each the same for all customers.

It is significant that the initial step using the $M/M/C/K$ model can be replaced by a corresponding initial step using the $M/GI/C/K + GI$ model, having IID service times with a non-exponential distribution (the first GI) and IID patience times with a non-exponential distribution (the $+GI$), drawing on the approximation algorithm in Whitt (2005). Moreover, we anticipate that the overall performance will be close to that predicted by the $M/GI/C/K + GI$ model, thus that single-class model can be used to predict and quantify the performance impact of these additional features in the SBR call center. For example, we could predict the consequence of the shift from exponential to hyperexponential service times in (8.1) above.

Other Sharing Schemes. Our algorithm assigns secondary skills in a balanced way, according to the fair-assignment rule in (5.3), but the algorithm should perform well with other assignments, providing that there is indeed adequate sharing across all workgroups. Evidently what is needed is something like the chaining specified by Jordan and Graves (1995). Accordingly, we performed analogs of the resource-pooling experiment in Section 4 in which the secondary skills were assigned by chaining. In particular, for $1 \leq i \leq 5$, all agents in workgroup i (with primary skill i) were assigned secondary skill $i + 1$, and all agents in workgroup 6 were assigned secondary skill 1.

The story in Section 4 is essentially the same with this less-balanced form of sharing, being only slightly worse. That is in distinct contrast to the much worse performance that occurs if, for all i, $1 \leq i \leq 3$, all agents in workgroup $2i - 1$ have secondary skill $2i$, while all agents in workgroup $2i$ have secondary skill $2i - 1$; then the entire call center performs as three separate call centers, each of size 30. That causes a serious performance degradation.

Different Numbers of Skills. We were naturally interested to see what happens when even less sharing is done. Thus we performed experiments in which only some of the agents have two skills, while the remaining agents have only a single skill. In particular, we considered variations of the resource-pooling experiment in Section 4 in which only 5 and 10 agents in each workgroup (of size 15) have secondary skills, with a balanced distribution among the five other workgroups. As expected, performance degrades as the degree of sharing decreases, but not proportionally. For example, the case of 5 yields a performance about equally between the case of 1 skill and 2 skills (instead of 1/3 of the way).

When the agents have different numbers of skills, new issues arise. To achieve better performance (but not necessarily to balance workloads), it tends to be advantageous to *route to the less flexible agents first*, if there is a choice. Once agents have different numbers of skills, attention needs to be paid to the *utilizations* of the different agents. For example, we found that when we applied our algorithm directly to these partial-skill cases, in the case of normal loading (as in Section 4), the less flexible agents (with only a single skill) had about twice the idle time as the more flexible agents (with 2 skills). When we routed calls to the less flexible agents first, the performance improved slightly and the utilizations became more balanced. Thus routing to the less flexible agents first was beneficial from both points of view. However, we have not yet done enough experiments to draw strong general conclusions. For any contemplated scenario, we would suggest paying attention to agent utilizations as well as the performance constraints in (2.1) and (2.2).

Percentage Served by Agents with Primary Skill. When calls may be served by agents with different skill levels, it is appropriate to pay attention to how many calls are served by agents with that call type as a primary skill, a secondary skill, and so on. We believe that is an important aspect of call-center performance. Accordingly, it is useful to report those statistics, e.g., as we did in Table 3.

9. Acknowledgments

This paper is based on the first author's doctoral dissertation at George Washington University, co-advised by Thomas A. Mazzuchi from George Washington University, William A. Massey from Princeton University and the second author, Ward Whitt from Columbia University.

References

Aksin, O. Z., F. Karaesman. 2002. Designing flexibility: characterizing the value of cross-training practices. INSEAD and Laboratoire Productique Logistique, Ecole Centrale Paris.

Anton, J., V. Bapat, B. Hall. 1999. *Call Center Performance Enhancement Using Simulation and Modelling.* Purdue University Press.

Armony, M., A. Mandelbaum. 2004. Design, staffing and control of large service systems: The case of a single customer class and multiple server types. New York University and the Technion. Available at: http://iew3.technion.ac.il/serveng/References/InvertedV.pdf

Azar, Y., A. Broder, A. Karlin, E. Upfal. 1994. Balanced allocations. *Proceedings of the 26th ACM Symposium on the Theory of Computing (STOC)* 593–602.

Bassamboo, A., A. Zeevi, J. M. Harrison. 2004. Design and control of a large call center: asymptotic analysis of an LP-based method. Stanford University and Columbia University.

Borst, S., P. Seri. 2000. Robust Algorithms for Sharing Agents with Multiple Skills. Bell Labs, Lucent Technologies. Unpublished report.

Brigandi, A.,D. Dargon, M. Sheehan, T. Spencer, III. 1994. AT&T's call processing simulator (CAPS): operational design for inbound call centers. *Interfaces* **24**, 6–28.

Brown, L., N. Gans, A. Mandelbaum, A. Sakov, H. Shen, S. Zeltyn, L. Zhao. 2002. Statistical analysis of a telephone call center: a queueing-science perspective. Department of Statistics, The Wharton School, University of Pennsylvania, Philadelphia, PA. To appear in *J. Amer. Stat. Assoc.*.

Chevalier, P., R. A. Shumsky, N. Tabordon. 2004. Routing and staffing in large call centers with specialized and fully flexible servers. Universite catholique de Louvain, University of Rochester and Belgacom Mobile/Proximus.

Choudhury, G. L., K. K. Leung, W. Whitt. 1995. Efficiently providing multiple grades of service with protection against overloads in shared resources. *AT&T Technical Journal*, **74**, 50–63.

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

Garnett, O., A. Mandelbaum. 2000. An introduction to skills-based routing and its operational complexities. Technion, Israel. Available at: http://iew3.technion.ac.il/serveng

Garnett, O., A. Mandelbaum, M. I. Reiman. 2002. Designing a call center with impatient customers. *Manufacturing and Service Opns. Mgmt.*, **4**, 208–227.

Green, L. 1985. A queueing system with general-use and limited-use servers. *Operation Research* **33**, 168–182.

Gurumurthi, S., S. Benjaafar. 2004. Modeling and analysis of flexible queueing systems. *Naval Res. Logistics*, to appear.

Halfin, S., W. Whitt. 1981. Heavy-traffic limits for queues with many exponential servers. *Operations Research* **29**, 567–588.

Harrison, J. M., A. Zeevi. 2003. A method for staffing large call centers based on stochastic fluid models. Stanford University and Columbia University.

Hopp, W. J., M. P. Van Oyen. 2003. Agile workforce evaluation: a framework for cross training and coordination. Northwestern University.

Jordan, W. C., S. C. Graves. 1995. Principles on the benefits of manufacturing process flexibility. *Management Science* 41, 577–594.

Jordan, W. C., R. R. Inman, D. E. Blumenfeld. 2003. Chained cross-training of workers for robust performance. General Motors Corporation.

Koole, G. M., J. Talim. 2000. Exponential approximation of multi-skill call centers architecture. *Proceedings of QNETs 2000*, Ilkley, UK, **23**, 1–10.

Mandelbaum, A., M. I. Reiman. 1998. On pooling in queueing networks. *Management Science* **44**, 971–981.

Massey, W. A., R. B. Wallace. 2005. An optimal design of the $M/M/C/K$ queue for call centers. *Queueing Systems*, to appear.

Mitzenmacher, M. 1996. *The Power Of Two Choices In Randomized Land Balancing.* Ph.D. Dissertation, University of California at Berkeley.

Mitzenmacher, M., B. Vöcking. 1999. The asymptotics of selecting the shortest of two, improved. *Proceedings of the 1999 Allerton Conference on Communication, Control and Computing*, University of Illinois.

Perry, M., A. Nilsson. 1992. Performance modeling of automatic call distributors: assignable grade of service staffing. *Proceedings of the 14th International Switching Symposium*, 294–298.

Puhalskii, A. A., M. I. Reiman. 2000. The multiclass GI/PH/N queue in the Halfin-Whitt regime. *Adv. Appl. Prob.* 32, 564-595.

Ridley, A. 2003. *Performance Analysis of a Multi-Class Preemptive Priority Call Center with Time Varying Arrivals*, Ph.D. dissertation, University of Maryland, College Park.

Stanford, D., W. K. Grassmann. 1993. The bilingual server systems: a queueing model featuring fully and partially qualified servers. *Management Science* **31**, 221–277.

Stanford D., W. K. Grassmann. 2000. Bilingual server call centers. *Analysis of Communication Networks: call centers, traffic and performance*, eds. D. McDonald and S. R. E. Turner (*Amer. Math. Soc.*, Providence), 31–47.

Turner, S. 1996. *Resource Pooling In Stochastic Networks*, Ph.D Dissertation, Cambridge University.

Turner, S. (1998). The effect of increasing routing choice on resource pooling. *Probability in the Engineering and Informational Sciences.* **12**, 109–124.

Vvedenskaya, N. D., R. L. Dobrushin, F. I. Karpelovich. 1996. Queueing systems with selection of the shortest of two queues: an asymptotic approach. *Problems in Information Transmission* **32**, 15–27.

Wallace, R. B. 2004. *Performance Modeling and Design of Call Centers with Skill-Based Routing*, Ph.D. dissertation, the George Washington University, School of Engineering and Applied Science.

Wallace, R. B., W. Whitt. 2004. A staffing algorithm for call centers with skill-based routing: supplementary material. Available at http://columbia.edu/∼ww2040.

Whitt, W. 1992. Understanding the efficiency of multi-server service systems. *Management Science*, **38**, 708–723.

Whitt, W. 2005. Engineering solution of a basic call-center model. *Management Science*, **51**, 221–235.

Wolff, R. W. 1989. *Stochastic Modeling and the Theory of Queues*, Prentice Hall, Englewood Cliffs, NJ.