Optimal Staffing of Systems with Skills-Based-Routing (SBR)

Zohar Feldman

Optimal Staffing of Systems with Skills-Based-Routing

Research Thesis

Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Operations Research and System Analysis

Zohar Feldman

Submitted to the Senate of the Technion - Israel Institute of Technology

Heshvan, 5769 Haifa Nov, 2008

This Research Thesis was conducted under the supervision of Professor Avishai Mandelbaum in the Faculty of Industrial Engineering and Management. I am deeply grateful to him for introducing me to the fascinating world of research, and for all of the many things that he taught me in the process. Working with him has been a great privilege and a truly exciting experience.

I would also like to thank Professor Arkadi Nemirovski from the Georgia Institute of Technology for his generous help and useful advice.

To my dear family and my precious wife, Lilach.
Your love and support are invaluable to me!

Finally, The Generous Financial Help Of The Technion Is Gratefully Acknowledged.

Contents

List of Figures

List of Tables

Abstract

In this work we optimize operational costs of systems with *Skills-Based-Routing* (SBR). In such systems, heterogeneous customers are routed to different types of servers, based on the servers' skills. In the settings we consider, each server skill is associated with a corresponding cost, and service level can either appear as a strong constraint or incur a cost.
This work is motivated mainly by Telephone Call Centers, where the term SBR is taken from. The SBR framework, however, arises in other contexts of service systems, for example technical support providers, banking systems, health-care etc. The solution we propose is based on the *Stochastic Approximation* (SA) approach. Since SBR models are analytically intractable in general, we use computer simulation to evaluate service-level measures. Under the assumption of convexity of the service-level functions in staffing levels, SA provides an analytical proof of convergence, together with a rate of convergence. We show via numerical examples that, although the convexity assumption does not hold for all cases and all types of service-level objectives, the algorithm still succeeds in identifying the optimal solution. We also apply our algorithms within time-varying environments, in particular to a realistic Call Center.

List of Symbols

λ_i Arrival rate of Customer Class i

$\mathcal{D}_i^P$ Patience distribution of Customer Class i

$\mathcal{D}_{ij}^S$ Service time distribution of Customer Class i by Servers Pool j

$X \sim \mathcal{D}$ X has distribution $\mathcal{D}$

N_j Number of servers in Server Pool j

$N_{j,t}$ The number of server in Server Pool j at time interval t

N Staffing levels vector $N = (N_1, \ldots, N_J)$

$\mathbb{P}_N \{A\}$ Probability of the event A under staffing levels N

T Number of iterations

γ Step-size

$\Pi_X (x)$ Projection of the point x onto the set X

$[x]^+$ Positive part of x, i.e. the maximum between x and 0

$\lceil x \rceil$ Smallest integer that is larger than x

$\partial f (x)$ Sub-differential of the function f - the set of all sub-gradients of f at point x

$P_X (y)$ Prox mapping

$V (x, y)$ Prox function

$d (x)$ Distance generating function

List of Acronyms

SBR Skills Based Routing

ISA Iterative Staffing Algorithm

LP Linear Programming

QIR Queue-Idleness-Ratio

FQR Fixed-Queue-Ratio

SP Static Priority

TP Threshold Priority

FSF Fastest Server First

SA Stochastic Approximation

SAA Sample-Average Approximation

SL Service Level

SRS Square-Root Staffing

1 Introduction

Call centers, and service systems in general, give rise to many operational problems, one of which is the *Staffing Problem*: under an existing operational reality, finding the minimal-cost staffing levels that is required to meet some given *Quality of Service* (QoS) constraints. This problem has received a great deal of attention over the years, as it rightly deserves: staffing costs are estimated at about 70% of a call center's operational costs. Staffing "wisely" can thus result in substantial savings while achieving operational objectives.

In the modern call center scene, customers are distinguished by the type of service they require. For instance, customers might be associated with different priority levels (VIP vs. Members) or different functional requirements (technical support vs. billing). Naturally, call centers address this situation by employing various types of servers, with varying sets of skills. This multi-class customers , multi-type servers system is often referred to as a system with *Skills Based Routing* (SBR) (See [7]). Due to model complexities, there exists only limited amount of exact analysis for specific SBR designs while many works address the general model heuristically. The staffing problem then becomes an integral part of the general problem of designing a call center with SBR. The three major problems when implementing SBR systems are: **design** (determine server types), **staffing** (determine staffing level of each type) and **control** (determine how customers are routed to servers). Although interrelated, these problems are typically addressed separately because of the complexity involved in addressing all three jointly.

In our research, we focus on the problem of staffing. Specifically, we rigorously relate the staffing problem of an SBR system to the formal framework of Stochastic Approximation (SA), as proposed in Juditsky, Lan, Nemirovski and Shapiro [15].

This relation enables the solution of SBR staffing, in the following sense:

1. An SA-based algorithm is proposed for a solution.
2. Under appropriate theoretical conditions, the algorithm converges at rates that were established in [15].
3. In practical conditions, an implementation of the algorithm yields solutions to two versions of the staffing problem: **cost optimization** and **constraint satisfaction**.

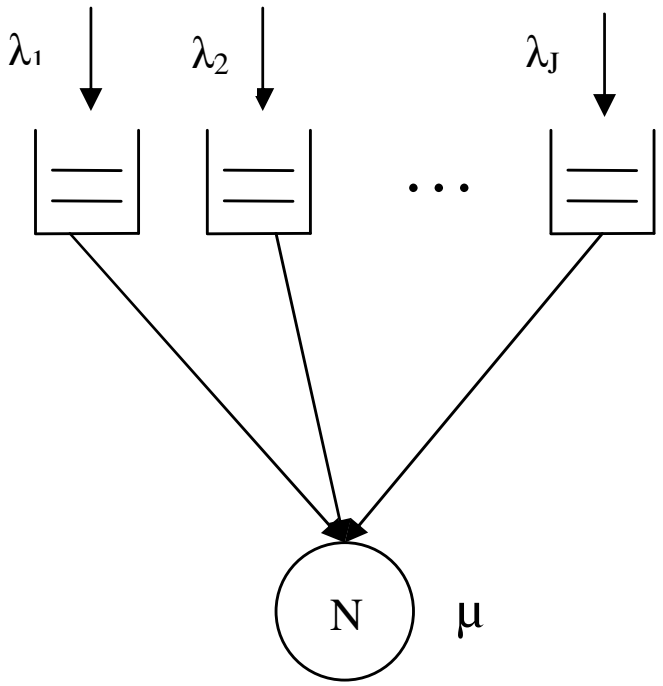
1.1 Literature Review

The constantly increasing body of literature on SBR systems includes exact analysis as well as asymptotic and heuristic analysis.
Gans, Koole and Mandelbaum [7] review the state of research on *Telephone Call Centers*. They explore the operational aspects and complexities associated with Call centers, including forecasting, time-varying arrival rates, uncertain arrival rates, staff scheduling and rostering, SBR and more. They also survey academic research devoted to the management of Call Centers operations, and outline important problems that have not been addressed.
A more recent survey is Aksin, Armony and Mehrotra [1]. Here the focus is on the multi-disciplinary nature of operational problems that arise in Call Centers.
In what follows, we mention only a selective set of related research that affected this current work, either in defining the research problem or in drawing ideas to solve it. For additional details, readers are referred to the surveys [7, 1].

Gurvich, Armony and Mandelbaum [11] explores the $\bigvee$-design, in which there are J customer classes, each having Poisson arrivals at rate $\lambda_j, j = 1, \ldots, J$ and N statistically identical servers, all capable of serving all classes of customers with a common service-time distribution $exp\,(\mu)$.

In this work, Gurvich et al. address the following joint problem of staffing and control: maximize profit, subject to *Quality of Service* (QoS) constraints that are expressed as a uniform upper bound on the fraction of class i customers who wait more than T_i units of time before starting service. There are also customer classes that do not have a service level constraint associated with them: they are referred to as best effort (and WLOG there is only one such class which is class J). Assume WLOG that $T_1 < T_2 < \ldots < T_{J-1}$, so that the lower the index the stricter is the constraint. In addition, another global constraint is imposed on the *Average Speed of Answer* (ASA) of the entire customer population. Gurvich et al, characterize the routing and staffing schemes that, under some assumptions, are asymptotically optimal in the limit, as the arrival rate and the number of servers increase to infinity. The suggested staffing rule is the *Single Class Staffing* (SCS) rule; namely, finding the number of agents required to satisfy the global ASA constraint

Figure 1: The $\bigvee$-design - Multiple Customer Classes and a Single Server Type



approximately, using a simple $M/M/N$ model. The corresponding control policy is the *Idle Server Threshold Priority* (ITP) scheduling: it assigns the head-of-the-queue class i customer upon customer arrival or service completion to an idle server if and only if:

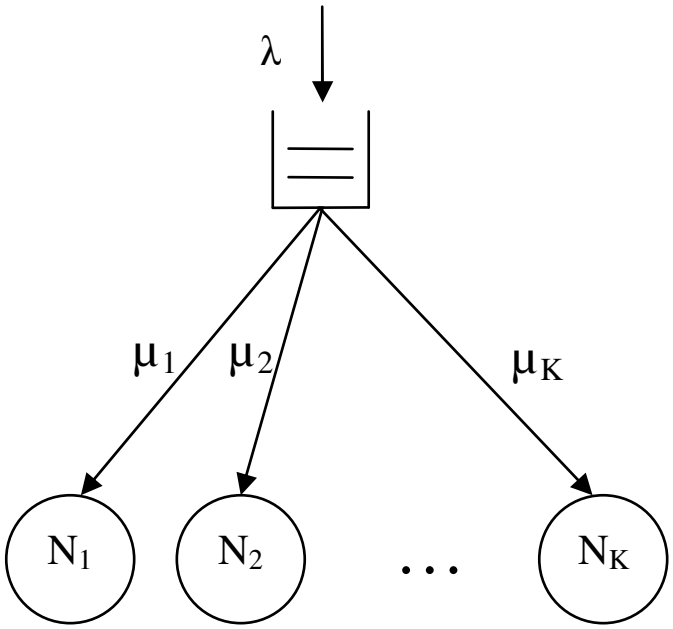
1. Queue j is empty for all higher priority classes j ($\forall j \leq i$) (those with stricter constraints on their service levels)
2. The number of idle servers exceeds a threshold K_i, where $0 = K_1 \leq K_2 \leq \ldots \leq K_J$

Once the number of agents N is determined using SCS, the thresholds $\{K_i\}$ are calculated using a recursive formula.

Armony and Mandelbaum [2] provides exact and asymptotic analysis of the inverted-$\bigvee$ design (also denoted $\bigwedge$-design), in which there is a single customer class and K server skills, each skill k has N_k servers in its pool, $k = 1, \ldots, K$; See Figure 2. Service times are assumed to be exponential, where the service rate μ_k depends on the skill type k of the particular server. Assume WLOG that that $\mu_1 < \mu_2 < \ldots < \mu_K$. Customers arrive

according to a Poisson process with arrival rate λ.

Figure 2: The $\bigwedge$-design - Single Customer Class and Multiple Server Skills



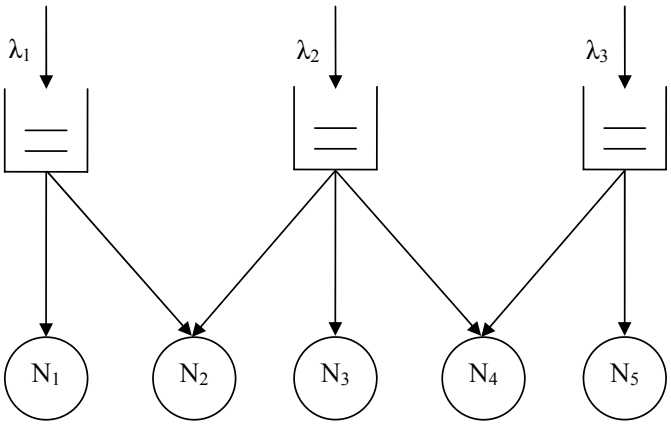
Armony and Mandelbaum explores the following joint problem of staffing and control: minimize staffing costs of the form $C_p(N) = c_1 \cdot N_1^p + \ldots + c_K \cdot N_K^p$, where $N = (N_1, N_2, \ldots, N_K)$ is the staffing level vector, subject to the constraint that the fraction of delayed customer should not exceed some given threshold α.

Regardless of the staffing costs, the asymptotic optimal control is identified as the *Fastest Server First* rule (FSF), assigning any newly arriving or waiting customers to the fastest available server. Indeed, separation of the control from staffing is a central finding in [2]. For $p > 1$, the optimal staffing level is proportional to the vector $\left(\frac{\mu_1}{c_1}, \frac{\mu_2}{c_2}, \ldots, \frac{\mu_K}{c_K}\right)$ and is determined by

$$\sum_{k=1}^{K} N_k \mu_k = \lambda + \delta\sqrt{\lambda}, \tag{1.1}$$

where $\delta = \beta\sqrt{\mu_1}$, β deduced from the *Halfin-Whitt* function $\alpha = \left[1 + \beta \cdot \frac{\phi(\beta)}{\varphi(\beta)}\right]^{-1}$, and μ_1 is the service rate of the *slowest* server skill. This formulae is in some sense the analog of the SRS for the $\bigwedge$-design. Note that in case $p = 1$, i.e. linear staffing costs, the control is to choose a single pool having the largest value of μ_i/c_i if it is also the globally slowest

Figure 3: The Generalized M Model For 3 Classes



type; otherwise it is unclear what to do.

Gurvich and Whitt, in [12] and [13], propose a family of Routing controls called Queue-Idleness-Ratio (QIR) rules. A special case of these rules is the Fixed-Queue-Ratio (FQR) rule, According to which, a newly available agent next serves the customer from the head of the queue of the class whose queue length most exceeds a specified proportion of the total queue length. Similarly, an arriving customer is routed to a pool of servers in which the number of idle servers most exceeds a specified proportion of the total number of idle servers. Under some regularity conditions on the network structure, [12] shows that FQR gives rise to state-space collapse which facilitates the establishment of asymptotical results. In simplified but natural settings, FQR is shown to provide a solution to the joint problem of design, staffing and routing in a nearly optimal manner. Starting with the design phase, the general model is reduced to a concatenation of M systems (termed the generalized M model), establishing a diminishing returns property, i.e. moderate cross-training is sufficient to make the call center work as efficiently as a single-pool system. See Figure 3 for an example of a generalized M Model with three customer classes. In the staffing phase, an aggregate approach is carried out to determine the total number of required servers. LP is then used to allocate the total number of servers to service pools, each with a designated skill set.

In addition to analytical and asymptotic analysis, there is an extensive amount of work that tries to address the staffing problem heuristically, and/or with the help of simulation. Wallace and Whitt [19] present a heuristic for staffing call centers with SBR. The following settings are considered: there are n customer classes, C servers, all serving customers according to their (not necessarily identical) set of skills, but all at rate μ independently of the customer type; finally, there are K extra waiting spaces. One major result demonstrated in [19] is that **little flexibility goes a long way**: with a limited amount of cross-training, well-designed SBR can perform nearly the same as if all agents have all skills. (This result was supported later on by the mathematical analysis in Bassamboo et al. [4]). For example, their experiments with simulation revealed that the performance of a system in which agents have 2 skills improve tremendously compare to a system in which all agents have only a single skill. However, the contribution of adding a skill for all agents becomes rather negligible beyond the second skill. Wallace and Whitt [19] apply the limited flexibility principle in order to develop a full algorithm to both route and staff call centers with SBR. Their objective is to find the minimal values of C and K, subject to some per-class QoS constraints of two types:

1. The blocking probability of type-k customer should not exceed ϵ_k, $k = 1, \ldots, n$.
2. The conditional probability that type-k customer waits more than τ_k time units, given that he was not blocked, should not exceed δ_k, $k = 1, \ldots, n$.

There is also a constraint in [19] on the number of skills that each agent enjoys. The general scheme of the solution is given as follows:
The algorithm in [19] finds initial values for C and K using the M/M/c/k model, with the total aggregate arrival rate $\lambda = \lambda_1 + \cdots + \lambda_n$. Then, the skills and priority levels for the C agents are determined. First, the number of agents C_k having a skill k as their first priority is determined for all $k = 1, \ldots, n$, so that $\sum_{k=1}^{n} C_k = C$ and $C_k = R_k + \beta \cdot \sqrt{R_k}$, where R_k is the per-class offered load. Second, the number of agents $C_{k,i}$ having skill k as a primary skill and skill i as their secondary skill is obtained using $C_{k,i} = \frac{C_k \cdot C_i}{C - C_k}$. The lower priority skills of each agent is chosen in a more or less arbitrary manner. At the proceeding steps, a feasible solution is obtained by increasing C and K and checking the feasibility of the solution using simulation: first, C is iteratively incremented, setting the skills and priorities according to the extent of constraint violation, i.e. the first priority

skill is the skill for which the difference between the target and service level is the highest, the second priority skill is the skill with the second highest value and so on. Once C is found, K and C are incremented until a feasible solution satisfying both constraints is obtained. At last, a refinement stage is applied to improve the solution by performing local search.

The proposed control is *Static Priority* (SP) defined by the prioritized skills of the servers: arriving calls of class k are routed to a server that has a primary skill k and has been idle for the longest time since service completion. If all agents in this group are occupied, the customer is routed to the group of agents whose skill k is their secondary skill. Again, a server with the longest idle time is chosen and so on. If none of the servers are available, the customer is queued. When a server becomes idle, he visits the queues of all classes feasible for his skills in the order of the agent's priority levels. Wallace and Whitt [19] shows by simulation experiments, that their joint solution for routing and staffing is nearly the same as if all agents had all skills and therefore it is near optimal.

Additional relevant works include [3], which identifies an optimal shift plan satisfying some given service level constraints. Namely, [3] finds the required number of servers in each shift, given a set of admissible shifts by applying the Cutting-Plane method and integer programming, where the relevant service level is calculated by simulation instead of analytical expressions.

1.2 Thesis Outline

In this work, we find optimal staffing levels in systems with SBR. In our settings, heterogenous customers are routed to appropriate skilled server according to some pre-specified routing control.

We define two different optimization formulations. In one, we find the minimal cost staffing levels whereas service level appears as a hard constraint that must be satisfied. In the other formulation, service levels incur costs. We thus try to identify the staffing levels that minimize the total operational costs.

Due to the analytical complexities of SBR models, it is impossible to calculate service levels for any configuration and staffing levels. Instead, we use simulation to evaluate

these values. The greatest advantage of using simulation is its ability to support complex models that capture reality far better than simplifying analytic models. Complicated features such as time-varying rates, general distributions and even uncertainty of parameter values can be accommodated in simulation quite easily.
On the other hand, the advantage of analytical models over simulation is their ability to provide insights as they typically provide closed-form expressions for the measures under consideration as function of the model parameters and the decision variables. Attaining a solution can therefore be immediate, or at least much faster than using simulation.
We use the Stochastic Approximation (SA) approach (as in [15]) as the core mechanism of our staffing algorithms. The SA approach is based on a Monte-Carlo sampling technique mimicking sub-gradient descent methods. To this end, we must make the assumption that the service level functions that we consider are convex in the staffing levels. An alternative approach, that was traditionally considered to produce superior performance over the SA method, is called the Sample Average Approximation approach (SAA). SAA approximates the stochastic programming function by averaging generated samples and performing some numerical procedure. However, in a recent work by Juditsky, Lan, Nemirovski and Shapiro [15], a modified SA is introduced and shown to be competitive and even outperform SAA.

This work is organized as follows. In §2 we introduce our model for an SBR service system and formulate corresponding optimal Staffing problems. We also describe in general the SA approach with some modified and improved procedures that we use in our algorithms. We describe in details our proposed SA-based solution in §3, presenting our algorithms and sub-routines. We also provide a proof of convergence stemming from Stochastic Approximation theory, and propose ways to calculate the necessary inputs for the algorithms in order to guarantee desired accuracy. Although we do not provide theoretical proof for convexity in the general case, we characterize the conditions under which it is more likely to expect the service level functions to be convex. We also mention a family of SBR systems, in which the convexity assumption was proven for several SL functions.
In §4 we describe a numerical study and demonstrate the accuracy of our algorithm in various settings of SBR systems and routing controls and also in time-varying environments. Our experiments show that the convexity assumption is very reasonable in many cases and, more importantly, even when the convexity assumption is refuted, our algorithms

still provide optimal solutions.

Finally, in §5 we describe a simulation tool for systems with Skills-Based-Routing, as used in this work. We also provide a short manual for a Staffing Optimizer GUI, which can be used for running simulation and optimizing SBR models.

2 The Staffing Problem

2.1 Model Formulation

The model we consider may be given in its most general form by the following description: there is a set $\mathcal{I}$ of customer classes, and a set $\mathcal{J}$ of server pools (stations). We denote the cardinality of the sets $\mathcal{I}$ and $\mathcal{J}$ by I and J, respectively. Customers of class $i \in \mathcal{I}$ arrive according to a Poisson process with rate λ_i , and the time they are willing to wait before abandoning is taken from a distribution $\mathcal{D}_i^P$ (the superscript P stands for Patience). Servers of pool $j \in \mathcal{J}$ process customers of class $i \in \mathcal{I}$ according to a service duration from distribution $\mathcal{D}_{i,j}^S$ (the superscript S stands for Service). Moreover, each pool of servers is associated with a cost such that the staffing cost per time unit of a server from pool $j \in \mathcal{J}$ is c_j.

We consider two optimization problems: the **Cost Optimization Problem** and the **Constraint Satisfaction Problem**.

The **Cost Optimization Problem** is formulated as follows:

$$\begin{aligned} \min_{N} \quad & c^T N + \sum_{k=1}^{K} f^k(N) \\ \text{s.t.} \quad & A^T N \leq b \\ & N \in \mathbb{Z}_+^J \end{aligned} \tag{2.2}$$

Here $f^k(N)$ are service-level related average cost functions, where cost is measured per unit of time. For instance, $f^k(N) = c_k^{ab} \lambda_k \mathbb{P}_N \{ab_k\}$ is the average cost of abandonments per unit of time, in case each abandonment of class k customer incurs a cost of c_k^{ab}. Another example for such a function may be $f^k(N) = \lambda_k \mathbb{E}_N \left[c_k^W (W_k)\right]$, which is the expected cost of waiting time per time unit, given a cost function c_k^W for the waiting-time-in-queue

of class k customers.

In the **Constraint Satisfaction Problem**, one is interested in solving

$$\begin{array}{ll} \min\limits_{N} & c^T N \\ \text{s.t.} & f^k(N) \leq \alpha_k \quad \forall k = 1, \ldots, K \\ & A^T N \leq b \\ & N \in \mathbb{Z}_+^J \end{array} . \tag{2.3}$$

Here $f^k(N)$ denotes a *Service Level Objective* associated with a specific performance measure and class of customers. The most common objective takes the form "no more than 20% of VIP customers are to wait more than 20 seconds" which can be expressed mathematically by $\mathbb{P}\{W_{VIP} > 20sec\} \leq 0.2$. Other typical constraints include performance measures such as the probability of abandonment ($\mathbb{P}\{Abandon\}$) or average wait ($\mathbb{E}[W_q]$). An important performance measure, rarely tracked, is the delay probability ($\mathbb{P}\{W_q > 0\}$).

In both problem formulations, we are allowing additional linear constraints on the staffing levels. These constraints arise from operational issues such as servers' availability, training considerations etc. For instance, if there are only 50 expert servers, one should incorporate the simple constraint $N_{expert} \leq 50$. The requirement that at least 10% out of the entire workforce should be trainees is translated into the linear constraint $N_{trainee} \geq 0.1\,(N_{trainee} + N_{expert})$, and so on. More importantly, we use these linear constraints to introduce Stability Conditions: in a multi-class, multi-pool system, this is the set of necessary conditions for stabilizing the system, i.e. none of the queues explodes in the long run. The contribution of these constraints is two-fold: first, the search space is reduced and thus convergence is faster; second, the service level functions, reduced to the restricted region, are more likely to be convex and thus the validity of the algorithm is guaranteed. Indeed, if we look at a single-class single-pool queue (G/G/N), then service level measurements such as the delay probability are convex on the set $N \geq R \triangleq \frac{\lambda}{\mu}$, but not on $\mathbb{Z}_+^J$. Namely, the delay probability is essentially 1 for all staffing levels that are smaller than the offered load R, and start decreasing in a convex manner from that point and on. In our multi-class, multi-pool situation, we expect to have the same behavior component-wise, i.e. if we examine the staffing levels in each pool while fixing the staffing in all others, the delay probability of a certain class will equal to one for any staffing levels

that falls out of the "stability" region, and decrease monotonically when increasing the staffing in the specific pool in the "stability" region. We provide examples in §4.

The main difficulty of solving (2.2) and (2.3) is the intractability of the functions f^k. Indeed, only for a scanty set of relatively simple models, with a specific design and control, only few objectives similar to the ones described above can be calculated either analytically, numerically or asymptotically. One is then led naturally to using simulation.

The solution we propose is based on a *Stochastic Optimization* approach called *Stochastic Approximation* (SA). We give a general outline of this approach and demonstrate its application to our problem in the following section.

2.2 Stochastic Approximation

Our discussion follows the notations and assumptions in Juditsky, Lan, Nemirovski and Shapiro [15].

The *Stochastic Approximation* (SA) approach uses *Monte-Carlo* sampling techniques to solve optimization problems of the form

$$\min_{x \in X} \{f(x) := \mathbb{E}[F(x, \xi)]\}, \tag{2.4}$$

where $X \subset \mathbb{R}^n$ is a non-empty bounded closed convex set, ξ is a random vector whose probability distribution $\mathbf{P}$ is supported on set $\Xi \subset \mathbb{R}^d$ and $F : X \times \Xi \to \mathbb{R}$. It is further assumed that for every $\xi \in \Xi$ the function $F(\cdot, \xi)$ is convex and that for every $x \in X$, the function $F(x, \cdot)$ is integrable with respect to $\mathbf{P}$.

The SA makes the following assumptions

Assumption 1. There is a way of generating iid samples $\xi_1, \xi_2, \ldots$ of the random vector ξ

Assumption 2. There is an Oracle at our disposal that, for any $\xi \in \Xi$ and $x \in X$, returns the value of $F(x, \xi)$ and a stochastic sub-gradient - a vector $\mathrm{G}(x, \xi)$ such that $g(x) := \mathbb{E}[\mathrm{G}(x, \xi)]$ is well defined and is a sub-gradient of $f(\cdot)$ at x, i.e. $f(y) \geq f(x) + g^T(x)(y - x), \forall y \in X$ $(g(x) \in \partial f(x))$.

Throughout this section we use the notation $\Pi_X(x)$ to denote the metric projection operator onto the set X, that is $\Pi_X(x) := \arg\min_{x' \in X} ||x' - x||$.

The *Classical SA* algorithm [18] solves (2.4) by mimicking the simplest sub-gradient descent method. The algorithm iteratively moves along the opposite direction of the generated sub-gradients with decreasing step-sizes to find the minimal valued solution. In some sense, the decreasing step-sizes are used for masking the "noise" of the sub-gradients, when reaching to the proximity of the optimal value.
The *Classic SA* is defined by the iterative step

$$x_{j+1} := \Pi_X \left(x_j - \gamma_j G \left(x_j, \xi_j \right) \right), j = 1, 2, \ldots \tag{2.5}$$

where, $\gamma_j = \frac{\theta}{j}$ is the step size and θ is calculated based on the strong convexity constant c of $f(x)$ (that is, $f(x') \geq f(x) + (x' - x)^T \nabla \frac{1}{2} c ||x' - x||_2^2, \forall x, x' \in X$). The expected error of the corresponding objective value is of order $O(j^{-1})$. Namely, $\mathbb{E}\left[f(x_j) - f(\bar{x}) \right] \leq \frac{L}{j}$ for some constant L and the minimizer $\bar{x}$. However, the algorithm parameters are based on knowing the value of the strong convexity parameter and performance is highly sensitive to it (see for example [15]).

The *Robust SA* [17], [15] does not make any assumptions regarding strong convexity. It maintains (2.5) intact:

$$x_{j+1} := \Pi_X \left(x_j - \gamma_j G \left(x_j, \xi_j \right) \right), j = 1, 2, \ldots \tag{2.6}$$

and it produces $\hat{x}_T$ as a *final solution*:

$$\hat{x}_T := \frac{1}{T} \sum_{t=1}^{T} x_t. \tag{2.7}$$

Here, $\gamma_j \equiv \frac{\theta}{\sqrt{T}}$ is constant. Namely, instead of using decreasing step-sizes, the *Robust SA* uses constant step sizes, and the final solution is the average over the trajectory $x_1, \ldots, x_T$[1]. The expected error of the objective is higher than the *Classis SA* ($O\left(T^{-0.5}\right)$), but the result is insensitive to the choice of θ and makes no further assumptions beyond convexity.

The *Mirror Descent SA* [15] is a generalization of the *Robust SA*. It is similar to the *Robust SA*, except that it uses a *Prox Mapping* $P_x(y)$ in the iterative step instead of (2.6). The Mirror Descent SA is defined by

$$x_{j+1} := P_{x_j} \left(\gamma_j G \left(x_j, \xi_j \right) \right), j = 1, 2, \ldots \tag{2.8}$$

$$\hat{x}_T := \frac{1}{T} \sum_{t=1}^{T} x_t, \tag{2.9}$$

[1] In some variants, only part of the trajectory is used for final the solution

where the *Prox Mapping* is given by

$$P_x(y) := \arg\min_{z \in X} \left\{ y^T (z - x) + V(x, z) \right\} \tag{2.10}$$

The function $V(x, z)$ from (2.10) is called the *Prox function* and is associated with a *distance generating function* $d : X \to \mathbb{R}$ as follows:

$$V(x, z) := d(z) - \left[d(x) + \nabla d(x)^T (z - x) \right]. \tag{2.11}$$

We do not go into details regarding the algebraic properties that the *distance generating function* d must satisfy. The interested reader is referred to [15] for further details. We only mention that the choice of a good *distance generating function* depends on the geometry of the problem. Note that in the case $d(x) := \frac{1}{2}||x||_2^2$, the procedure (2.8) coincides with (2.6). Indeed, it is easy to see that for $d(x) := \frac{1}{2}||x||_2^2$, we have that $P_x(y) = \Pi_X(x - y)$. The expected error of the objective is again $O(T^{-0.5})$.

Juditsky et al. [15] also shows how the *Mirror Descent SA* algorithm can be modified to solve the convex-concave saddle point problem

$$\min_{x \in X} \max_{y \in Y} \{ \phi(x, y) := \mathbb{E}[\Phi(x, y, \xi)] \}, \tag{2.12}$$

where $\phi : X \times Y \times \Xi \to \mathbb{R}$ is convex in $x \in X$ and concave in $y \in Y$ for every $\xi \in \Xi$.
A special case of (2.12) is the Minimax problem taking the following form:

$$\min_{x \in X} \max_{k=1,\ldots,K} \left\{ f^k(x) := \mathbb{E}\left[F^k(x, \xi) \right] \right\}, \tag{2.13}$$

which is exactly the same as solving the Saddle Point Problem

$$\min_{x \in X} \max_{y \in Y} \left\{ \phi(x, y) := \sum_{k=1}^{K} y_k f^k(x) \right\}. \tag{2.14}$$

Here $Y = \left\{ y \in \mathbb{R}^K : \sum_{k=1}^K y_k = 1, y \geq 0 \right\}$ is the standard k-dimensional simplex.
The solution to (2.14) is given by the *Mirror SA for the Saddle Point Problem*. Let $z := (x, y)$ and $Z := X \times Y$, and let $\mathrm{G}(z, \xi) := \mathrm{G}(x, y, \xi)$ be the stochastic sub-gradient

$$\mathrm{G}(x, y, \xi) = \begin{bmatrix} \mathrm{G}_x(x, y, \xi) \\ -\mathrm{G}_y(x, y, \xi) \end{bmatrix}, \tag{2.15}$$

such that $\mathrm{g}_x(x, y) = \mathbb{E}[\mathrm{G}_x(x, y, \xi)] \in \partial_x \phi(x, y)$ and $-\mathrm{g}_y(x, y) = -\mathbb{E}[\mathrm{G}_y(x, y, \xi)] \in \partial_y(-\phi(x, y))$.
Then the *Saddle Point Mirror SA* algorithm is defined by the iterative step

$$z_{j+1} := P_{z_j}(\gamma_j \mathrm{G}(z_j, \xi_j)), \tag{2.16}$$

and the final solution

$$\hat{z}_T := \left(\sum_{t=1}^{T} \gamma_t\right)^{-1} \sum_{t=1}^{T} \gamma_t z_t. \tag{2.17}$$

In (2.16), γ_j is the step-size; the *Prox mapping* $P_z(\zeta)$ is defined similarly to (2.8), and associated with the *distance generating function* $d : Z \to \mathbb{R}$, which is in turn defined in terms of the distance generating functions $d_x(x)$ and $d_y(y)$ in the following way

$$d(z) := \frac{d_x(x)}{2D_{d_x,X}^2} + \frac{d_y(y)}{2D_{d_y,Y}^2}, \tag{2.18}$$

with $D_{d,X} := [\max_{z\in X} d(z) - \min_{z\in X} d(z)]^{1/2}$. More specifically, for the Saddle Point Problem (2.14), a simple derivation yields the stochastic sub-gradient

$$\mathrm{G}(x, y, \xi) := \begin{bmatrix} \sum_{k=1}^{K} y_k \mathrm{G}^k(x, \xi) \\ \left(-F^1(x, \xi), \ldots, -F^K(x, \xi)\right) \end{bmatrix} \tag{2.19}$$

3 The Staffing Algorithms

First, we establish the connection between the objective functions in (2.4) and (2.13) considered by the SA and the Service Level functions from our optimization problem (2.2) and (2.3).

Denote by $(\Omega, \mathcal{F}, P)$ the probability space formed by the distributions of arrival times, service times and patience. For some SL types, the expectation form arises quite naturally. For instance, if we look at SL functions of the form $f(N) = \mathbb{E}[\#Abandons]$, i.e. the expected number of abandonments, over a given optimization horizon and staffing levels $N \in \mathbb{R}^n$, then we fall exactly into the domain of SA. Namely, define $F : \mathbb{R}^J \times \Omega \to \mathbb{R}$ such that $F(N, \omega)$ is a random variable denoting the number of abandonments, given a possible outcome ω and staffing levels $N \in \mathbb{Z}_+^J$. Obviously, we can write

$$f(N) := \int_{\Omega} F(N, \omega)\, d\mathbb{P}(\omega), \tag{3.20}$$

which is equivalent to $f(x) := \mathbb{E}[F(x, \xi)]$ in (2.4): in (2.4) the random variable $F(x, \omega)$ is defined with an intermediate random vector ξ, where in our case it is imbedded in the probability space $(\Omega, \mathcal{F}, P)$.

In other cases, however, we may need to perform additional steps before using SA.

For example, when using simulation, the conventional way to represent the SL function $\mathbb{P}\{W > \tau\}$ is by the following ratio:

$$\mathbb{P}_N\{W > 0\} = \frac{\int_\Omega D(N,\omega)\, d\mathbb{P}(\omega)}{\int_\Omega A(\omega)\, d\mathbb{P}(\omega)}; \tag{3.21}$$

here $D(N,\omega)$ denotes the number of delayed customers and $A(\omega)$ denotes the total number of arrivals, over a given time-horizon.

Fortunately, the expectation in the denominator of (3.21) is independent of the staffing levels N. We can thus accommodate such SL functions as follows: In Cost Optimization, f appears in the objective function. Since the denominator is independent of the decision variables N, we can ignore it completely as it has no affect over the solution. In this case we just set $F(N,\omega) := D(N,\omega)$.

In the Constraint Satisfaction, the SL functions appear as constraints, e.g.

$$\mathbb{P}\{W > \tau\} \leq \alpha \tag{3.22}$$

In this case, we translate (3.22) into

$$\mathbb{E}[D(N,\cdot)] \leq \alpha \mathbb{E}[A(\cdot)], \tag{3.23}$$

where $\mathbb{E}[A(\cdot)]$ can be calculated in advance, and therefore the right-hand-side of (3.23) is constant. We explain later in 3.2 how we deal with constraints in our proposed algorithm.

Theoretically, ω here may be an infinite-size vector as it may include an infinite number of arrivals. In practice, however, when we seek to estimate steady-state service levels, we are looking at a finite number of arrivals, starting from a point in which we believe the system is stable. In other cases, we are interested in transient service levels. For example, in time-varying situations, we may wish to look at the the delay probability during each hour. We thus generate a sample path of the system only at the relevant time.

We use simulation for systems with SBR to both generate iid sample $\omega_1, \omega_2, \ldots$ from Ω, and calculate the values of $F^k(N,\omega)$ for any $N \in \mathbb{R}_+^J$ and $\omega \in \Omega$ that arise along the steps of the algorithms. An important point to be made here is that in practice, we are required to evaluate the functions $F^k(N,\omega)$ for non-integral values of N, even though N is discrete in our case (there is no such thing as a fraction of a server). In this case, we round the point N to the nearest integral point and plug it into the simulation to obtain the required value.

Similarly to [3], we evaluate the stochastic sub-gradient $\mathrm{G}^k(N,\omega)$ by taking finite differences of length 1 as follows:

$$\left[\mathrm{G}^k(N,\omega)\right]_i := F^k(N+e_i,\omega) - F^k(N,\omega). \tag{3.24}$$

Here, e_i denotes a vector of the same dimension as N, in which all entries are zeros except for the i'th component which equals to one.
There are several optional ways of producing sub-gradients, such as infinitesimal perturbation analysis [20] and likelihood ratio gradient estimation [10]. In fact, the finite differences method does not guarantee to yield a sub-gradient. Examples in which this method fails are easy to find. Nevertheless, finite differences still appear as a reasonable method and our numerical studies demonstrate that the method could yield optimal solutions.

Recall from Section 2.2 that the SA depends strongly on the convexity of the objective function $f(x)$ defined in (2.4). We are therefore required to make the same assumption on our SL functions $f^k(N) = \mathbb{E}\left[F^k(N,\cdot)\right]$ defined in (2.3) and (2.2):

Assumption 3. [Convexity of service level objectives] $\forall\omega\in\Omega$, the functions $F^k(\cdot,\omega)$ are convex, and for every $N\in\mathbb{Z}_+^J$, $F^k(N,\cdot)$ are integrable, with respect to P.

Remark 3.1. we are not trying to say, and it is evidently untrue, that all the types of SL functions that we consider are convex for any configuration of design and routing scheme. However, there exist supporting theory for the convexity of some performance measures under some configurations. Moreover, it appears that even in the case that Assumption 3 fails to hold, the obtained solution is still satisfactory and even sometimes optimal. For instance, functions that turn out to be convex on the feasible region but not on the whole space. We discuss this further in §3.4

Recall from Section 2.2 and [15] that SA provides computational procedures for calculating the optimal solution of two types of optimization problems:

1. The minimization problem $\min_{x\in X}\mathbb{E}\left[F(x,\xi)\right]$
2. The Minimax problem $\min_{x\in X}\max_{k=1,\ldots,K}\mathbb{E}\left[F^k(x,\xi)\right]$.

The Cost Optimization Problem (2.2) conforms exactly to Formulation 1 of the minimization problem. However, in the Constraints Satisfaction Problem (2.3) the SL functions appear as constraints rather than objectives. To resolve this, we make use of the Minimax problem in the following manner:

- There exist a feasible solution[2] that incurs a cost C if and only if the following condition holds
$$\left(\min_{x\in X_C}\max_{k=1,\ldots,K}\left\{f^k(x)-\alpha_k\right\}\right)\leq 0,$$
where $X_C := \left\{x\in X : c^T x = C\right\}$
- We search for the minimal C for which we can find a feasible solution in an efficient way such as a binary search

We elaborate and describe in details both the Cost Optimization and Constraints Satisfaction algorithms in the following sections.

3.1 Cost Optimization

We apply the Robust SA algorithm in a straightforward manner to solve the Cost Optimization Algorithm.
Few preliminary calculations are required in order to provide the step size γ and the number of iteration T, which guarantee a desirable accuracy of the solution. First, we set the search space X to be the intersection of the polyhedron $\left\{x : A^T x \leq b\right\}$ from (2.2) with the J-dimensional cube $\mathcal{C} = \left\{x : 0 \leq x_j \leq x_j^b \quad \forall j = 1,\ldots,J\right\}$. Here, x_j^b is the required number of agents in pool j that guarantees a perfect service experience (delay probability of all the customer classes that are served by this pool is essentially 0), given that all customers are served by this pool only. We compute the value of x_j^b by finding the minimal staffing level in a corresponding $M/M/N$ system for which the delay probability is 0 (practically, smaller than some small threshold) with arrival rate $\lambda = \sum_{i\in I(j)}\lambda_i$ and service rate $\mu = \frac{1}{\lambda}\sum_{i\in I(j)}\frac{\lambda_i}{\mu_{ji}}$; $I(j)$ denotes the set of classes that are served by pool j. Note that any solution that is not in $\mathcal{C}$ is dominated in terms of cost by some solution in $\mathcal{C}$. Indeed, by reducing the staffing levels of all the components to the level of the bound

[2] A *feasible solution* is a solution in which all SL constraints are satisfied

we decrease the staffing costs, but do not increase the cost of SL and therefore improve the overall cost. We choose the initial solution $x_1 := \Pi_X\left(\frac{1}{2}x^b\right)$.

Next, in order to determine the step-size γ and required number of iterations T, we must evaluate the term

$$M_*^2 := \sup_{x\in X} \mathbb{E}\left[||G(x,\cdot)||_2^2\right], \tag{3.25}$$

where G is the sub-gradient of the objective function, composed of the service cost vector and the summation of sub-gradients of the SL cost functions. Formally, $G(x,\omega) := c + \sum_{k=1}^{K} G^k(x,\omega)$. We propose a way to evaluate M_*^2 in §3.3.

Moreover, we calculate $D_X := \max_{x\in X} ||x - x_1||_2 = \frac{1}{2}||x^b||_2$.

Finally, the required number of iterations is given by

$$T := \left(\frac{D_x M_*}{\epsilon\delta}\right)^2 \tag{3.26}$$

and the step-size is

$$\gamma := \frac{D_x}{M_*\sqrt{T}}, \tag{3.27}$$

Table 1: Cost Optimization Algorithm

Initialization	Set $i := 1$;
Step 1	Run simulation to generate ω_i and obtain the value $G^k(x_i,\omega_i)\ \forall k = 1,\ldots,K$;
Step 2	Set $G(x_i,\omega_i) := c + \sum_{k=1}^{K} G^k(x_i,\omega_i)$, $x_{i+1} := \Pi_X(x_i - \gamma G(x_i,\omega_i))$;
Step 3	If $i \geq T$ return the solution $\hat{x}_T := \frac{1}{T}\sum_{t=1}^{T} x_t$;
Step 4	Set $i := i + 1$. Go to Step 1;

The Cost Optimization algorithm is displayed in Table 1.

Theorem 3.1. *Upon termination of the Cost Optimization Algorithm at point $\hat{x}_T$, with step-size γ, we achieve*

$$\mathbb{P}\left\{f(\hat{x}_T) - f(\bar{x}) \geq \delta\right\} \leq \epsilon, \tag{3.28}$$

where f is the objective function in the Cost Optimization Problem (2.2), and $\bar{x}$ is a minimizer of f.

Proof. It follows from (2.19) in [15] that $\mathbb{E}\left[f(\hat{x}_T) - f(\bar{x})\right] \leq \epsilon\delta$. Then (3.28) is immediate by the Markov inequality. □

3.2 Constraint Satisfaction

The Constraint Satisfaction Problem appears to be more complicated in the sense that it does not fit the form of the optimization problem (2.4). Namely the functions $f^k(x)$ appear as constraints rather than objective functions. For this matter, we start first with solving the following Minimax Problem

$$\min_{x \in X_C} \max_{k=1,\dots K} \left\{ f^k(x) - \alpha_k \right\}, \tag{3.29}$$

where $X_C := \left\{ x \in X : c^T x = C \right\}$. As already mentioned in (2.14), this is equivalent to solving the Saddle Point Problem

$$\min_{x \in X_C} \max_{y \in Y} \left\{ \sum_{k=1}^{K} y_k \left(f^k(x) - \alpha_k \right) \right\}. \tag{3.30}$$

Suppose that solving (3.29) yields some solution $\hat{x}$. We shall then be able to identify, with the help of some additional validation mechanism, if $\hat{x}$ satisfies the constraints in (2.3). Namely, there is some feasible solution x that incurs a cost of C monitory units, if and only if $\max_{k=1,\dots,K} \left\{ f^k(\hat{x}) - \alpha_k \right\} \leq 0$. We then search for the minimal cost C for which a feasible solution exists, in a binary search fashion, where the above procedure of solving (3.29) and validating the solution feasibility is used to either return the feasible solution or determine that such one does not exist.

We describe our validation mechanism $Feasible(x)$, which returns whether a solution x is feasible or not with confidence level α and accuracy δ, in 3.2.1.

We make additional efforts to transform X_C into a *standard simplex*. In this case, the choice of the *entropy-distance-generating-function* $d(x) := \sum_{i=1}^{n} x_i \ln x_i$, potentially yields optimal performance. (The error of the algorithm turns out almost independent of the number of constraints - $O\left(\sqrt{\ln K}\right)$, and the number of variables (pools) - $O\left(\sqrt{\ln J}\right)$).

To be able to work with the *standard simplex*, we define new variables $\tilde{x} := (\tilde{x}_1, \dots, \tilde{x}_J)$ by

$$[\tilde{x}]_i := \frac{x_i c_i}{C}, i = 1, \dots, J, \tag{3.31}$$

where c_i is the per-time staffing cost of pool i server, and C is the overall cost. Similarly, let $\tilde{z} := (\tilde{x}, y)$. For notational convenience, $x(\tilde{x})$ will actually stand for the vector x with components $x_i = \tilde{x}_i C / c_i$.

Our problem is now formulated by

$$\min_{\tilde{x}\in\tilde{X}}\max_{y\in Y}\left\{\sum_{k=1}^{K}y_k\left(f^k\left(x\left(\tilde{x}\right)\right)-\alpha_k\right)\right\}, \tag{3.32}$$

where both $\tilde{X}$ and Y are standard simplices with the dimensions J and K respectively. By choosing d_x and d_y to be the *entropy distance generating function*, it is easy to show that the *Prox Mapping* $P_z\left(\zeta\right)$ defined in (2.10) with the distance generating function d_z in (2.18) is given by

$$\left[P_z\left(\zeta\right)\right]_i=\begin{cases}\dfrac{z_iJ^{-2\zeta_i}}{\sum_{j=1}^{J}z_jJ^{-2\zeta_j}}, & \text{i=1,...,J;}\\ \dfrac{z_iK^{-2\zeta_i}}{\sum_{j=1}^{J}z_jK^{-2\zeta_j}}, & \text{i=J+1,...,J+K.}\end{cases} \tag{3.33}$$

Moreover,

$$\mathrm{G}_{\tilde{z}}\left(\tilde{z},\omega\right):=\begin{bmatrix}\sum_{k=1}^{K}y_k\mathrm{G}_{\tilde{x}}^k\left(x,\omega\right)\\ \left(-F^1\left(x,\omega\right)+\alpha_1,\ldots,-F^K\left(x,\omega\right)+\alpha_K\right)\end{bmatrix}, \tag{3.34}$$

where

$$\left[\mathrm{G}_{\tilde{x}}^k\left(x,\omega\right)\right]_i:=\frac{d}{dx_i}F^k\left(x,\omega\right)\frac{d}{d\tilde{x}_i}x_i=\left[\mathrm{G}_x^k\left(x,\omega\right)\right]_i\frac{C}{c_i} \tag{3.35}$$

We can now define the solution to (3.32) to be the outcome x^* of the $MirrorSaddleSA$:

$$\tilde{z}_{j+1}:=P_{\tilde{z}_j}\left(\gamma_j\mathrm{G}_{\tilde{z}}\left(\tilde{z}_j,\omega_j\right)\right) \tag{3.36}$$

$$\hat{z}_T:=\left(\hat{x}_T,\hat{y}_T\right)=\left(\sum_{t=1}^{T}\gamma_t\right)^{-1}\sum_{t=1}^{T}\gamma_t\tilde{z}_t \tag{3.37}$$

$$x^*:=x\left(\hat{x}_T\right) \tag{3.38}$$

with the constant step-size

$$\gamma_j:=\frac{2}{M_*\sqrt{5T}}. \tag{3.39}$$

Here,

$$M_*^2:=2M_{*,x}^2\ln J+2M_{*,y}^2\ln K \tag{3.40}$$

and $M_{*,x}^2,M_{*,y}^2$ are the bounds

$$\begin{aligned}&\mathbb{E}\left[\|\mathrm{G}_{\tilde{x}}^k\left(x,\cdot\right)\|_\infty^2\right]\le M_{*,x}^2\quad\forall k,x\\ &\mathbb{E}\left[F^k\left(x,\cdot\right)^2\right]\le M_{*,y}^2\quad\forall k,x\end{aligned}. \tag{3.41}$$

We are referring to the procedure (3.36)-(3.38) as the $MirrorSaddleSA$ associated with the cost C, or more specifically $MirrorSaddleSA(C)$.

Theorem 3.2. *Following the Mirror Saddle SA procedure (3.36)-(3.38) with the step size (3.39) we have that, for any $\delta > 0$,*

$$\mathbb{P}\left\{\max_{k=1...K} f^k(x^*) - \phi_* > \delta\right\} \leq 2M_*\sqrt{\frac{5}{T\delta^2}}, \tag{3.42}$$

where ϕ_* is the optimal solution to the Minimax problem 3.29

Proof. Immediate from (3.22) in [15] and the Markov inequality. □

Corollary 3.3. *It follows from Theorem 3.2 that in order to guarantee accuracy of δ with probability at least $1-\epsilon$, one can use the sample size*

$$T \geq \frac{20M_*^2}{\delta^2\epsilon^2}. \tag{3.43}$$

Discussion (accuracy of estimators)**.** In both the Cost Optimization and Constraint Satisfaction algorithms, we use $F^k(x,\omega)$ (resp. $\mathrm{G}^k(x,\omega)$) to estimate the actual value of the SL function $f^k(x)$ (resp. subgradient $g^k(x)$). $F^k(x,\omega)$ and $\mathrm{G}^k(x,\omega)$ are called *unbiased* estimators, since $f^k(x) = \mathbb{E}\left[F^k(x,\cdot)\right]$ and $g^k(x) = \mathbb{E}\left[\mathrm{G}^k(x,\cdot)\right]$.
Note that the only place where the variances $\mathbb{V}ar\left[F^k(x,\cdot)\right]$ and $\mathbb{V}ar\left[\mathrm{G}^k(x,\cdot)\right]$ come up is through the term M_*. In both cases, the required number of iterations T that guarantees a desired accuracy δ and confidence ϵ depends on a factor of M_*^2, which bounds the second moment of our estimators $F^k(x,\omega)$ and $\mathrm{G}^k(x,\omega)$ (See (3.41) and (3.25)) . Obviously, we can reduce the required number of algorithm iterations by reducing the variance of our estimators. This can be achieved by basing our estimators on a larger sample. In our case, larger sample means generating several simulation sample paths of the system on a given time-horizon, or using longer sample path (larger time-horizon), but this is valid only for steady-state estimation. However, increasing the sample size naturally increases the computational efforts. The important question to ask here is what is the optimal sample size that minimizes the overall computational efforts? Let us assume that producing our estimators is the most expensive in terms of computational effort, and that all other procedures are negligible compared to it. We also assume that the time required to produce an estimator is proportional to the sample size. We thus aim to minimize the required efforts associated with estimator calculation, by determining the optimal sample size. The total efforts required throughout the algorithm is of order $s \times M_*^2(s)$, where s is the sample size, and $M_*^2(s)$ is proportional to the required number of iterations. Since

$M_*^2(s)$ includes constants as well as terms that decrease at rate $\frac{1}{s}$, we conclude that the optimal sample size is the minimal one that provides unbiased estimators.

Next, we define the range of the cost in which we perform the binary search.
For this purpose, We calculate an upper bound for the optimal cost. Since we cannot assume anything regarding the routing scheme, we consider the case where each class of customers i is routed to pool j which maximizes the cost of the required staffing level N_{ij} for meeting the SL constraint. We can compute the required staffing levels for typical SL functions using a simple stationary queueing system (such as Erlang-C, or Erlang-A). Other SL functions that cannot be computed in this way, we replace with the constraint $\mathbb{P}\{W > 0\} = 0$ (practically, smaller than some threshold) which is the strongest constraint possible in the sense that satisfying the constraint $\mathbb{P}\{W > 0\} = 0$ implies satisfying any other SL constraint. Let

$$x_{ij} = \begin{cases} N_{ij}, & j \in \arg\max_k c_k N_{ik}; \\ 0, & \text{otherwise.} \end{cases}$$

and let $x_j = \sum_{i=1}^{I} x_{ij}$. Then our upper bound is given by $C_{max} = c^T x$.
Finally, the Constraints Satisfaction Algorithm is displayed in Table 2.

Table 2: Constraint Satisfaction Algorithm

Initialization $dC := C_{max}$, $x^* := x$;
Step 1 if $dC \leq \delta_C$ return the solution x^*;
Step 2 $dC := dC/2$;
Step 3 if $Feasible(x)$ returns true then $x^* := x$, $C := C - dC$, $x := x\frac{C}{(C+dC)}$. Go to Step 1;
Step 4 $x := MirrorSaddleSA(C)$;
Step 5 if $Feasible(x)$ returns true then $x^* := x$, $C := C - dC$, $x := x\frac{C}{(C+dC)}$ Go to Step 1;
Step 6 $C := C + dC$, $x := x\frac{C}{(C-dC)}$. Go to Step 1;

Theorem 3.4. *Following the constraint Satisfaction procedure, with accuracy of δ, cost accuracy δ_C and confidence level ϵ in the sub-procedures $Feasible$ and $MirrorSaddleSA$, we achieve the following:*

1. *Feasibility:* $\mathbb{P}\left\{\max_{k=1,\ldots,K}\left(f^k\left(\hat{x}\right)-\alpha_k\right)>\delta\right\}\leq 1-(1-\epsilon)^{\lceil\log\frac{C_{max}}{\delta_C}\rceil}$

2. *Optimality:* $\mathbb{P}\left\{c^T\hat{x}-c^*>\delta_C\right\}\leq 1-(1-2\epsilon)^{\lceil\log\frac{C_{max}}{\delta_C}\rceil}$

Proof. First, note that the CS algorithm terminates after exactly $\lceil\log\frac{C_{max}}{\delta_C}\rceil$ iterations. Denote by $\hat{x}_{(i)}$ the solution obtained by the i-th iteration. Then,

1. The probability that $Feasible\left(\hat{x}_{(i)}\right)$ returns the answer 'true' given that $\hat{x}_{(i)}$ is not feasible is bounded by ϵ. Hence, the probability that the final solution $\hat{x}$ is not feasible is bounded by the probability that in at least one iteration i of the algorithm, the sub-procedure Feasible wrongly declared $\hat{x}_{(i)}$ as feasible. Since there are $\log C_{max}$ iterations, we deduce the result.

2. The probability that at some iteration i with a feasible cost C_i[3], the algorithm will fail to identify a feasible solution is bounded by the probability that either the sub-procedure MirrorSaddleSA fails to find a δ-optimal solution or that it successfully finds such a solution but it is wrongly denied by the sub-procedure Feasible. The first event is bounded by ϵ, and the latter is bounded by $\epsilon\left(1-\epsilon\right)$. The overall bound is thus $2\epsilon-\epsilon^2<2\epsilon$. In the worst case, the cost will be feasible at all iterations and hence the result.

□

3.2.1 Solution Feasibility

Our algorithms use simulation in order to evaluate service level objectives. In this section we describe how we determine whether a certain solution N is feasible in the sense that it satisfies the SL constraints with some accuracy δ and level of confidence α. Assume we would like to estimate $f^k\left(N\right)=\mathbb{P}\left\{W_k>T\right\}$, where W_k denotes the waiting time of class k customers. For this purpose we run n independent simulation replications. Let A_i^k be the number of arrivals in replication i, and $D_i^k\left(N\right)$ is the number of class k customers

[3]A *feasible cost* C is a cost, for which there exists a δ-feasible solution $\hat{x}$, that is $\max_{k=1,\ldots,K} f^k\left(\hat{x}\right)-\alpha_k<\delta$.

which had to wait more than T time units. Then, we use the estimator

$$\hat{f}^k(n) := \frac{\sum_{i=1}^n D_i^k}{\sum_{i=1}^n A_i^k}. \tag{3.44}$$

Furthermore, the confidence interval for $\mathbb{P}\{W_k > T\}$ is given by $\left[lb^k(n), ub^k(n)\right]$, where

$$\begin{aligned} lb^k(n) &:= \hat{f}^k(n) - z_\alpha \sqrt{\frac{\sigma_D - 2\sigma_{D,A}\hat{f}^k(n) + \hat{f}^k(n)^2\sigma_A}{n\bar{A}^2}} \\ ub^k(n) &:= \hat{f}^k(n) + z_\alpha \sqrt{\frac{\sigma_D - 2\sigma_{D,A}\hat{f}^k(n) + \hat{f}^k(n)^2\sigma_A}{n\bar{A}^2}} \end{aligned} \tag{3.45}$$

In (3.45) σ_A and σ_D are the sample variance of A and D, respectively, $\sigma_{A,D}$ is the their sample covariance and z_α is the α-percentile of the *Standard Normal* distribution. To determine whether a solution N is feasible or not, we run simulation and stop at iteration n if and only if one of the following conditions hold:

$$\forall k : ub^k(n) \leq \alpha_k + \delta, \tag{3.46}$$

or

$$\exists k : lb^k(n) \geq \alpha_k + \delta. \tag{3.47}$$

In the first case we say that the solution is feasible, and in the latter it is not.

3.3 Calculating the Bound M_*^2

In order to calculate the the required number of iterations for both algorithms, we must evaluate the expression M_*^2 in (3.25) and (3.40). To do this, one must be able to obtain bounds for the estimator of the SL functions (resp. subgradient) $\mathbb{E}\left[F^k(N,\cdot)^2\right]$ (resp. $\mathbb{E}\left[G^k(N,\cdot)^2\right]$). In what follows, we propose a bound for the estimators of the SL function $f(N) := \mathbb{P}_N\{W > 0\}$. Other SL functions can be handled similarly. In fact, in some cases, the bound for the delay probability can serve as bound for other SL types (e.g. the probability to abandon, tail probability, etc.).

We start with $\mathbb{E}\left[F^k(N,\cdot)^2\right]$: recall that after we "translate" $\mathbb{P}\{W > 0\}$ to the division of the expected number of delayed arrivals $\mathbb{E}[D(N,\cdot)]$ by the expected number of arrivals $\mathbb{E}[A(\cdot)]$, we are actually considering only the nominator. Namely, we set $F(N,\xi) := D(N,\omega)$, and the components of the subgradient $G(N,\omega)$ symbolize the number of arrivals that are delayed in the original system, but admitted to service upon arrival

when there is an additional server in the respective pool. Obviously, $F(N,\omega) \leq A(\omega)$ since the number of delays up to a given time T cannot exceed the number of arrivals during that time. Note that $A(\omega) \sim Poisson(\lambda T)$, and therefore $\mathbb{E}\left[F(N,\omega)^2\right] \leq \lambda^2 T^2 + \lambda T$.

Next, we bound $\mathbb{E}\left[G^k(x,\cdot)^2\right]$. An immediate bound for $G(N,\omega)$ is the number of delays itself which is in turn bounded by the number of arrivals $A(\omega)$. Adding a server can reduce the number of delays of a certain class either directly in case the server can cater to the specific class or indirectly by diverting load from a pool that caters to that class. Let $S_{ij}(\omega)$ be the number of potential service completions of class i made by a server from pool j in T time units, i.e. when one service is starting immediately after the completion of a preceding service. It is easy to see that the reduced number of delays in class i by pool j is bounded by $S_{ij}(\omega)$ if pool j is serving class i. In this case $[G^i(N,\omega)]_j \leq \min\{A(\omega), S_{ij}(\omega)\}$, where $S_{ij}(N,\omega) \sim Poisson(\mu_{ij}T)$.
It is more likely that under a reasonable routing policy the affect of adding a server to a pool that serves class i would be higher than adding a server to a pool that doesn't serve class i. However, it is certainly possible that the case would be the other way around. For example, one can think of a system in which there is a pool that serves some class a and also serves some other class b with relatively long service times. Now let us suppose that there is an additional pool that serves class b, but much faster than the first pool. Adding a server in that pool might divert much of the load that is directed to the first pool. Consequently, servers of the first pool would be more available to serve class a, possibly much more than the contribution of an additional single server. We may still be able to bound the indirect contribution in the same spirit of the previous bound. However, calculations may get extremely complicated when considering the general case. We thus settle for the trivial bound $A(\omega)$.

3.4 Convexity

Intuitively, one expects that, with the proper representation, the service level functions $f^k(x)$ under consideration should be component-wise monotone decreasing and convex. That is, service level such as the probability $\mathbb{P}_N\{W_k > T_k\}$ is decreasing (not increasing)

as the number of agents in any pool increases for any reasonable routing sceme[4]. Moreover, the marginal decrease should get smaller as any component of N increases similarly to the behavior of the simple single-class single-pool queue (Erlang models).

Indeed, for this special case of SBR systems, with a single class of customers and a single pool of servers, there exists supporting theory for the convexity of several SL functions, such as the delay probability $\mathbb{P}_N\{W > 0\}$ or the tail probability $\mathbb{P}_N\{W > T\}$.

In this work, we are imposing a stronger assumption: the SL functions are *jointly convex* in N. This assumption need not prevail for any given SBR model, and in fact we show later in Section 4 examples where our assumption is refuted.

However, we propose a heuristic to characterize here the conditions under which the convexity assumption is plausible, relating it to stability consideration that naturally arise for the model under consideration.

It seems that what mostly affects the convexity of the service level functions is the underlying design of the model, and to some extent also the routing scheme.

We assert that whenever the necessary conditions for stability that we introduce below, expressed in linear constraint, form a convex space, it is likely that the service level function would be convex and vice versa. For instance, let us consider an arbitrary N-model (see Figure 4) with two classes of customers and two pools of servers. Let λ_1 and λ_2 be the the arrival rates of class 1 and class 2, respectively; pool 1 serves class 1 with rate μ_{11}; Pool 2 serves class 1 with rate μ_{21} and class 2 with rate μ_{22}.

To keep class 1 from exploding, one must require that the total service rate of pool 1 servers and pool 2 servers will be greater or equal to the arrival rate of class 1. This can be expressed mathematically by

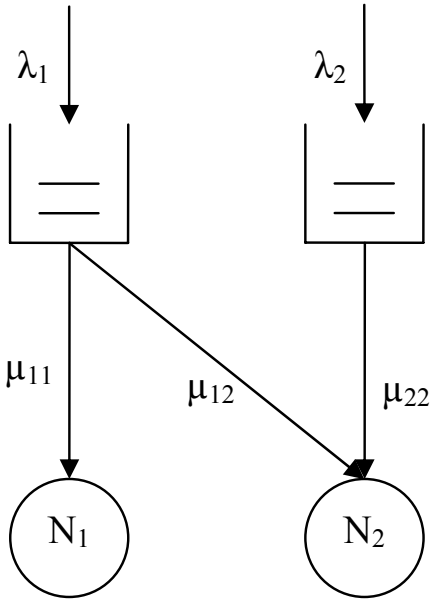
$$\left[N_2 - \frac{\lambda_2}{\mu_{22}}\right]^+ \mu_{21} + N_1\mu_{11} \geq \lambda_1 \tag{3.48}$$

It is easy to see that (3.48) comprise a non-convex space (see blue curve in Figure 5). Our numerical study demonstrates that in these cases the SL function is indeed not convex (See for example Figure 13)

Having said that, there is still room for optimism as even in cases in which not all the SL functions are convex on the entire space they may be convex within a region containing the

[4]One can cook up some pathological routing scheme, dependent of specific staffing levels, that negates the monotonicity and convexity conjecture, but these rules are impractical.

Figure 4: Arbitrary N Model

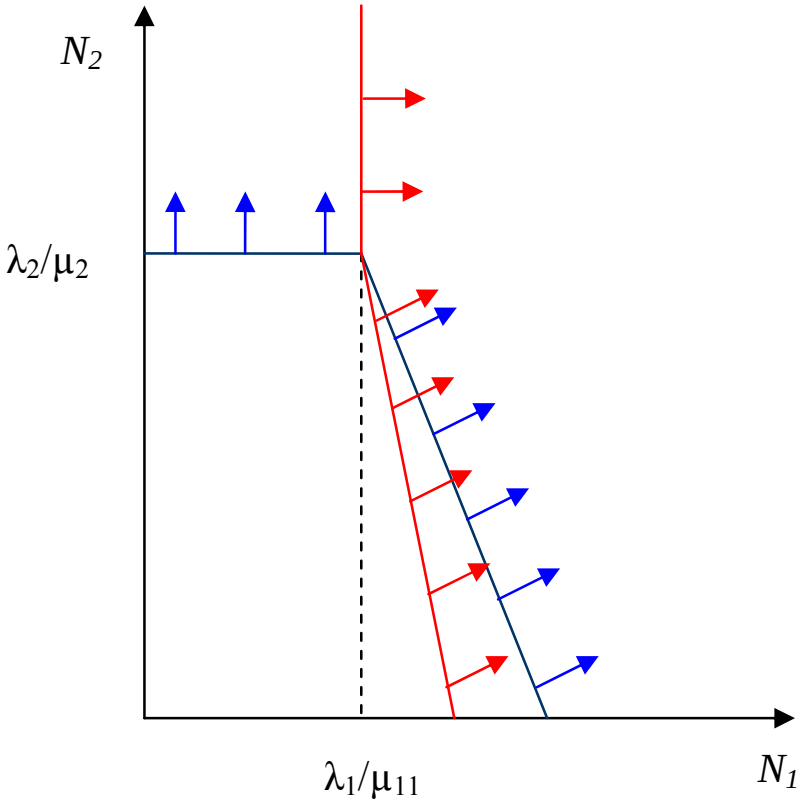


relevant points. For instance, if we add the stability condition for class 2 in our previous example

$$N_2\mu_{22} \geq \lambda_2 + [\lambda_1 - N_1\mu_{11}]^+, \tag{3.49}$$

then the formed space is convex (red curve in Figure 5). Going back to Figure 13, it is visible that the SL function of the second class indeed turns out to be convex. More importantly, it seems that on the restricted region, the service level functions of both classes are convex.

Figure 5: Stability Conditions



4 Experimental Results

In this section we walk through a simple example of a skills-based-routing system. We continuously modify the model configuration in order to test our algorithm performance in various settings. We use simulation to reconstruct the real functional constraints and consequently identify the real optimal solution. This provides the solid ground for comparing our results to the optimal solution. The main purpose of this numerical study is to show the accuracy of our algorithm. Moreover, we would like to show that the convexity assumption that we are making is actually valid in many cases, and in the cases that it is not valid we are still doing surprisingly well.

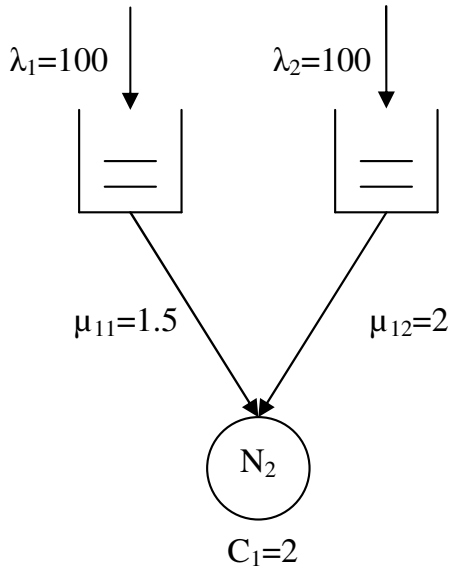
4.1 V Model

We start off with describing the general settings of the service system that we consider. We look into a situation in which there are two distinct classes of customers, say $class_1$ and $class_2$. Each one of the two classes is associated with a different service level agreement and may have different service requirements. The arrivals are assumed to be *Poisson* with class dependent arrival rate $\lambda_i, \quad i = 1, 2$. At first, we consider the case where the two classes are served by a single pool of servers. That is, all servers are statistically identical. This model is known as the V model arising from the shape of the corresponding graph representation of this system. For the sake of example, let us say that arrival rate for both classes is 100, that is $\lambda_1 = \lambda_2 = 100$. $class_1$ is associated with the constraint $\mathbb{P}\{W_1 > 0\} \leq 0.2$ and $class_2$ has the constraint $\mathbb{P}\{W_2 > 0\} \leq 0.5$. Moreover, each server processes customers of $class_1$ according to *exponential* distribution with rate $\mu_1 = 1$ and customers of $class_2$ with rate $\mu_2 = 1.5$.

As in [11], we use the *Threshold Priority* control in order to differentiate the service level of the two classes. Namely, we let $class_2$ customers to be admitted to service if"f there are no customers of $class_1$ waiting in queue, and there are at least k available servers. Finally, our goal is to find the optimal number of servers n^* and the control threshold k^* for which the service constraints are met.

Observation 1. For any V model with a Threshold Priority control, we can construct an equivalent N model with *Static Priority* control. In what follows, we show the construction

Figure 6: Schematic Representation of the V-model Settings



for the case of two classes. The construction for three and more classes follows the same lines. Let the number of agents in the V model be n, and let the threshold for class 2 be k. We construct the N model as follow: We put k servers in pool 1 and n-k servers in pool 2. The routing schema is as follows: arriving customer of $class_1$ is routed to pool 1 if there is available server in that pool. Otherwise he is routed to pool 2 or queued if there is no available server also in pool 2. Customers of $class_2$ can only be routed to pool 1. Upon service completion, servers of pool 1 will try to take any one of the customers that are in service in pool 2. If there are none they will look to serve customer from the head of $class_1$ queue and only then from $class_2$ queue. Servers of pool 2 will only look in the queue of $class_1$.

Having formalized the equivalence between the two models, we can apply our algorithm to solve the corresponding problem (4.50) with the equivalent N model, and derive the optimal solution by the simple transformation

$$(n^*, k^*) \leftrightarrow (N_1^* + N_2^*, N_2^*)\,.$$

For the sake of stability, we demand that $N_1 \geq 167$.

$$\begin{array}{ll} \min\limits_{N} & 1N_1 + 1N_2 \\ \text{s.t.} & \mathbb{P}_N\{W_1 > 0\} \leq 0.2 \\ & \mathbb{P}_N\{W_2 > 0\} \leq 0.5 \\ & -N_1 \leq -167 \\ & N \in \mathbb{Z}_+ \end{array} . \tag{4.50}$$

As it may be clearly viewed in Figure 7, both service level functions, reduced to the feasible area that is determined by the stability conditions, are convex in N. This guarantees the accuracy of the solution obtained by our algorithm. Indeed, the solution we obtained $N^* = (171, 1)$ turned out to be one of the optimizers of (4.50) as was verified by simulation (see the derived feasible region and optimal solution in Figure 8). Moreover, this results is consistent with the theoretical solution from [11].

4.2 N model

4.2.1 Static Priority control

In this example we take the same settings as in §4.1, only here we intend to use two pools of servers. $pool_1$ is dedicated only to $class_1$ and serves them with rate $\mu_{11} = 1$. $pool_2$ serves both classes with service rates $\mu_{21} = 1.5$ and $\mu_{22} = 2$ for $class_1$ and $class_2$ respectively. $pool_1$ are only half as costly as $pool_2$ as they are less skilled and less productive. Since there is no significance to the cost values rather to their proportions, let us say that each server from $pool_1$ costs 1 monetary units per time unit and each server from $pool_2$ costs only 2 monetary units per time unit. For routing we use the *Static Priority*: $class_1$ are having higher priority for $pool_1$. When becoming available, servers of $pool_2$ has higher priority to $class_1$. We first state the stability conditions

$$\begin{array}{l} N_1 + 1.5N_2 \geq 100 \\ 2N_2 \geq 100 \\ N_2 \geq \dfrac{100}{2} + \dfrac{100 - N_1}{1.5} \end{array} . \tag{4.51}$$

Figure 7: Performance Measures For V Model With *Threshold-Priority*

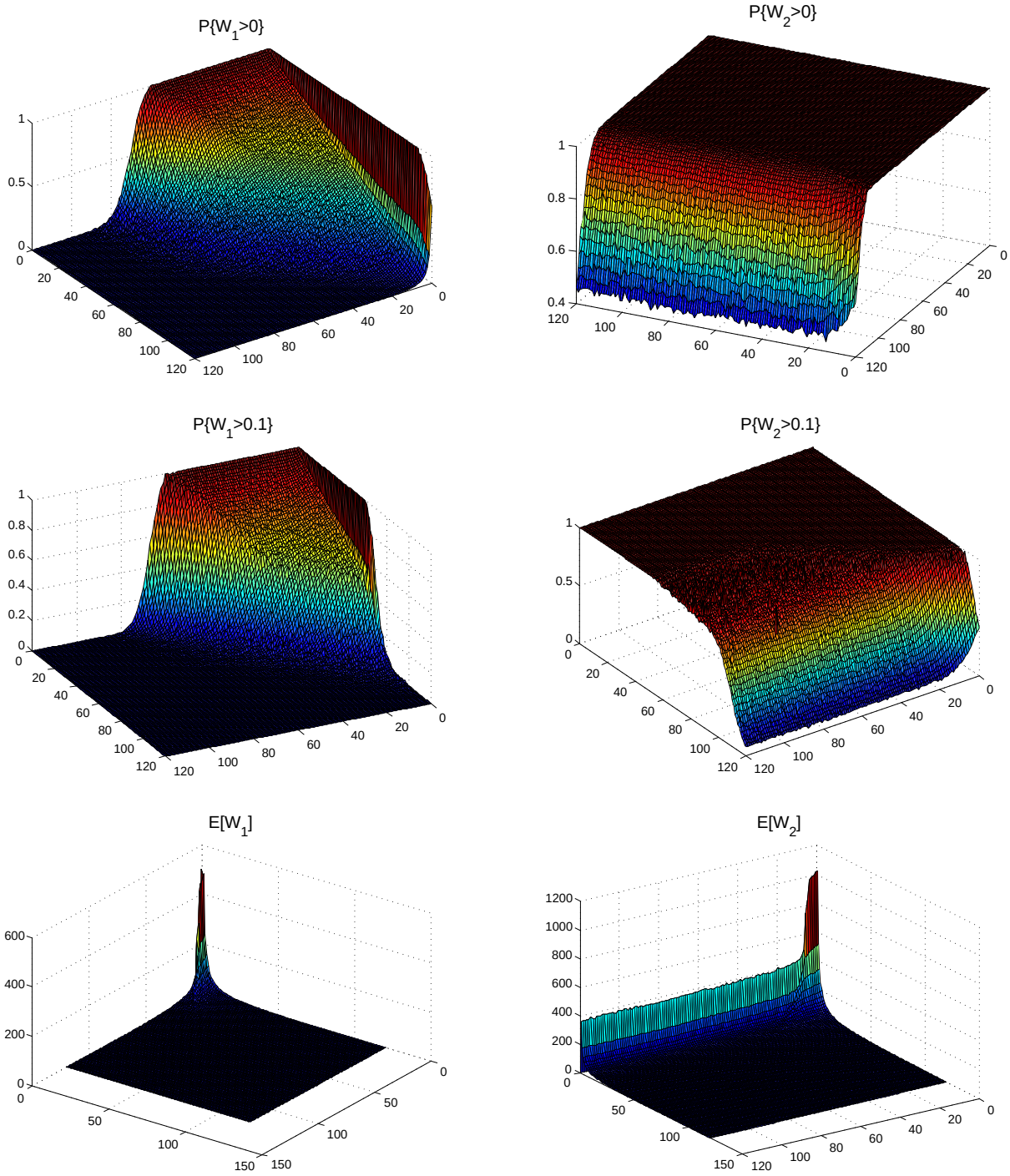


Figure 8: Feasible Region and Optimal Solution

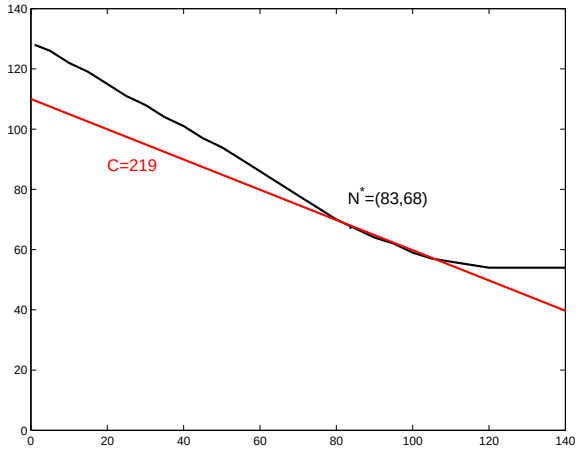
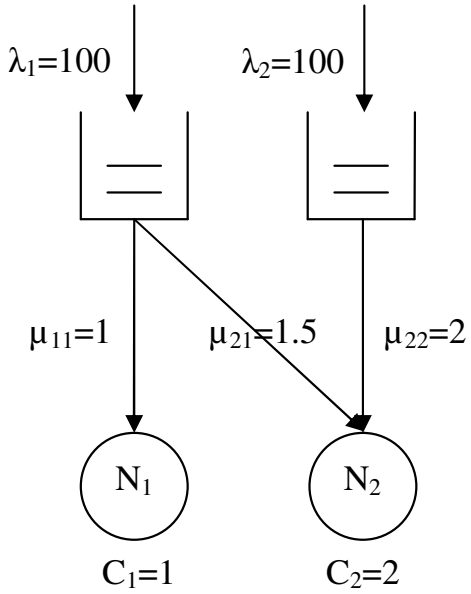


Figure 9: Schematic Representation of the N-model Settings



Since the first condition is redundant we set $A = \begin{bmatrix} 0 & -1 \\ -\frac{2}{3} & -1 \end{bmatrix}$, and $b = \begin{bmatrix} -50 \\ -117 \end{bmatrix}$. The problem is instantiated into

$$\begin{aligned} \min_{N} \quad & 1N_1 + 2N_2 \\ \text{s.t.} \quad & \mathbb{P}_N\{W_1 > 0\} \leq 0.2 \\ & \mathbb{P}_N\{W_2 > 0\} \leq 0.5 \\ & AN \leq b \\ & N \in \mathbb{Z}_+ \end{aligned} \quad . \tag{4.52}$$

We used simulation in order to obtain the values of the delay probability of both classes as function of the staffing levels. The results are displayed on Figure 10

From Figure 10 it is easy to derive the feasible area along with the optimal solution as shown in the left hand side of Figure 11. The solution we got was $N = (83, 68)$, which was in fact one of the optimizers of (4.52).

Let us now impose another restriction to our problem. On top of the functional constraints and the stability conditions in (4.52), we will also like to have that at least 50% of the overall staffing levels will be servers of $pool_2$, which is translated into the linear equation $N_2 \geq N_1$. Again, our algorithm succeeded to identify the optimal solution $N^* = (74, 75)$ (see right hand size of Figure 11.

Another interesting experiment is to associate a cost to the waiting times of customers, and try to solve a Cost Optimization formulation. Let us assume that a cost of one time unit of wait is 0.2 for $class_1$ customers, and 0.1 for $class_2$ customers. This is consistence with $class_1$ having a more strict service level constraint in the previous problem. In the problem formulation (4.56), we multiply the waiting costs per time unit by the arrival rate to represent the expected cost of waiting times.

$$\begin{aligned} \min_{N} \quad & 1N_1 + 2N_2 + 20\mathbb{E}_N[W_1] + 10\mathbb{E}_N[W_2] \\ \text{s.t.} \quad & AN \leq b \\ & N \in \mathbb{Z}_+ \end{aligned} \quad . \tag{4.53}$$

The optimal solution obtained by simulation was found to be $N^* = (80, 54)$, while our solution was $N = (88, 47)$ (only 2 cost units difference).

Figure 10: Performance Measures for N model Case With *Static Priority*

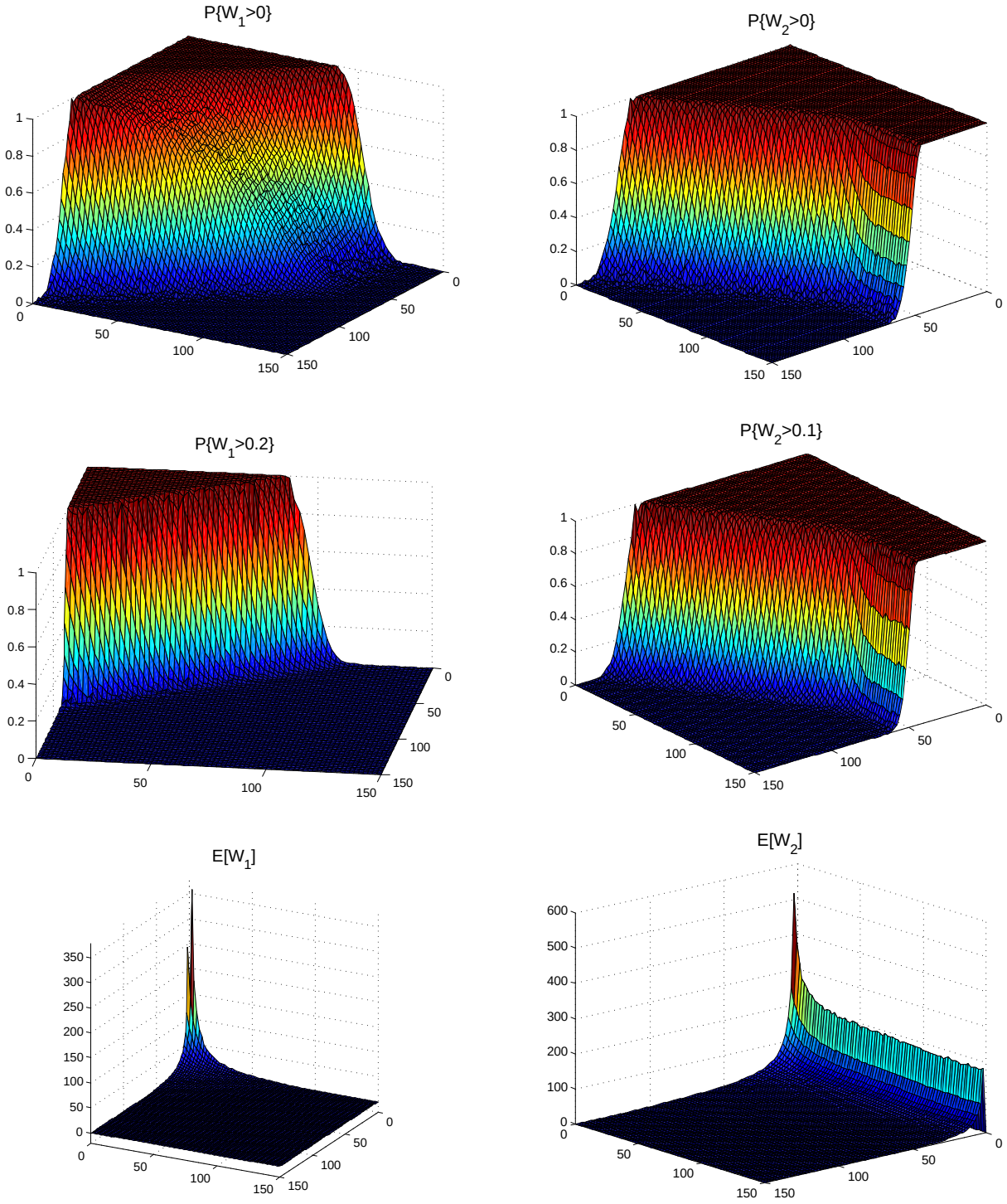


Figure 11: Feasible Region and Optimal Solution
basic setting with additional constraint

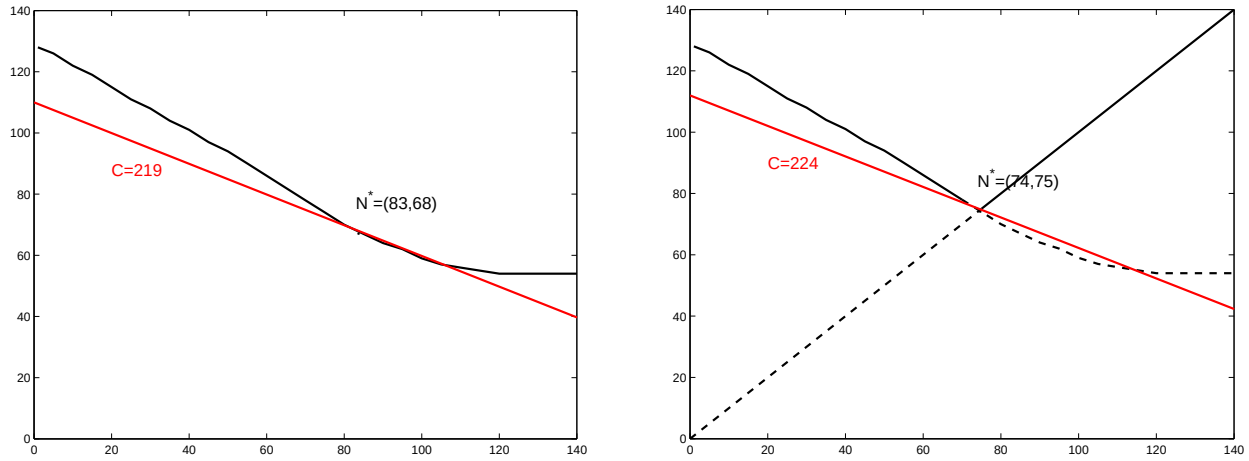
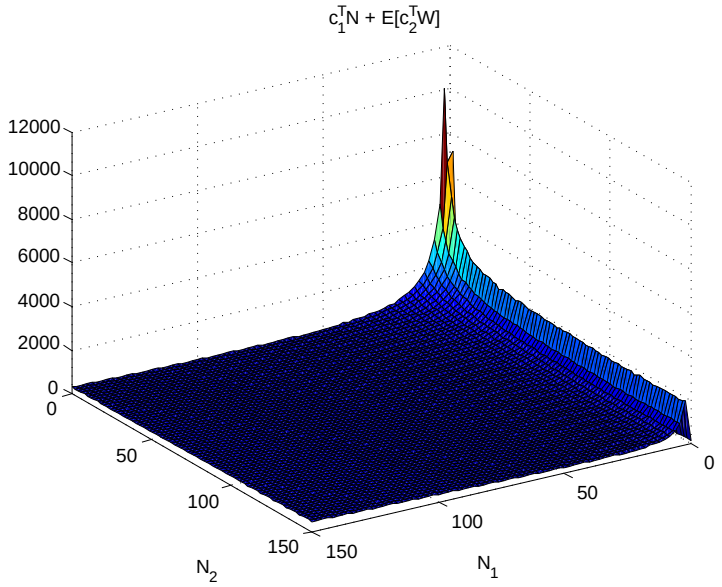


Figure 12: Cost Function for N model Optimization Formulation



4.2.2 FQR control

Now, we would like to change the service level constraints. We are now interested in the tail probability. We set the target for $class_1$ to be $T_1 = 0.1$ time units and for $class_2$ $T_2 = 0.2$, and we demand that not more than 20% of each class will have to wait more than their target. This is translated into

$$\begin{array}{ll} \min\limits_{N} & 1N_1 + 2N_2 \\ \text{s.t.} & \mathbb{P}_N\left\{W_1 > 0.1\right\} \leq 0.2 \\ & \mathbb{P}_N\left\{W_2 > 0.2\right\} \leq 0.2 \\ & AN \leq b \\ & N \in \mathbb{Z}_+ \end{array} \quad . \tag{4.54}$$

Again, although the tail probability turns out to be not convex, it is still convex on the feasible area. Again, We derive the feasible area and optimal solution by simulation. The solution we produced was again very close to optimal ($N = (91, 60)$ with a total cost of 211, compared to 210 which was the optimal cost in this case).

Interestingly, the routing scheme has a significant affect on the optimal solution. Had one used the Static Priority control for the same problem, it would have turned out that the optimal solution would be $N^* = (93, 64)$ (see right hand side of Figure 14) with a total cost of 221 monetary units, 11 units more than the cost when using FQR control. We could expect that FQR would achieve better performance, since it is shown in [12] to have good qualities with constraints of the tail probability type. SP, on the other hand, can be naive in the sense that it does not try to adjust to dynamics of the system. In other cases, we observed even higher differences.

4.2.3 Deterministic Service Times

Here we assume that the service times are deterministic, i.e. the time that it takes a server from pool j to serve a customer of class i is $d_{ji} = \frac{1}{\mu_{ji}}$. Consider the basic N model with static priority, and deterministic service times $d_{11} = 1$, $d_{21} = \frac{2}{3}$ and $d_{22} = 0.5$.

Figure 13: Performance Measures for N model Case With FQR

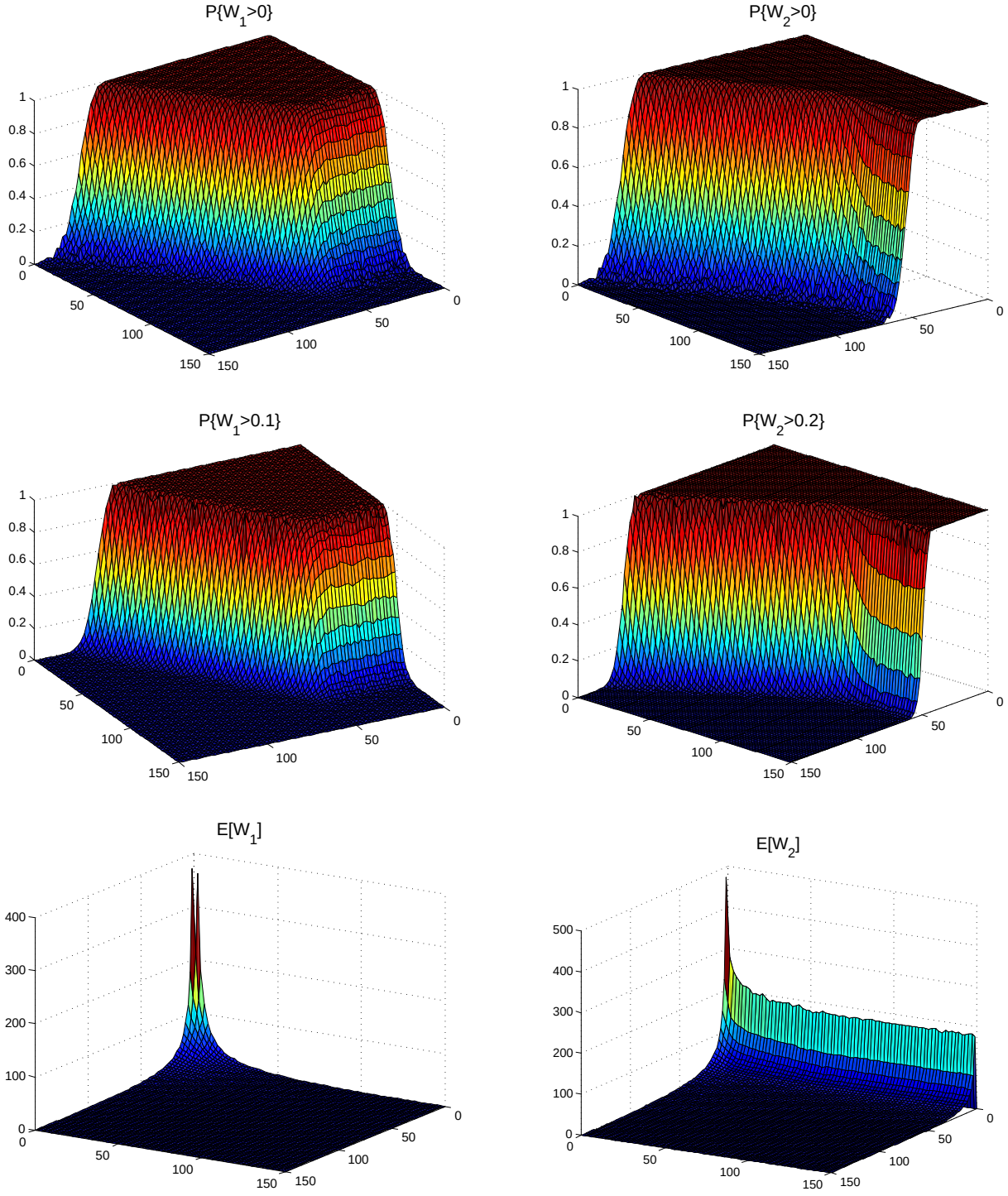
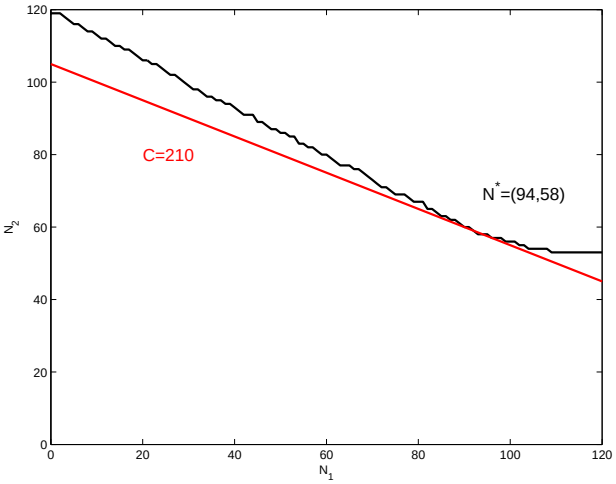
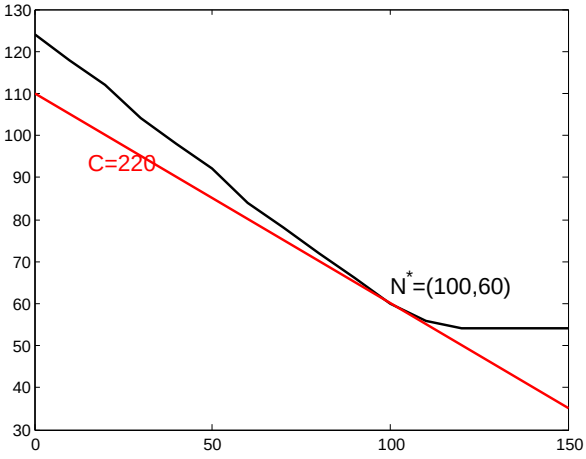


Figure 14: Feasible Region and Optimal Solution for Target Tail Probability

FQR control



SP control



4.2.4 Log-Normal Service Times

In most practical service systems, the Log-Normal distribution seems to fit the actual distribution of service times the most. The Log-Normal distribution has two parameters: the mean μ and the variance σ^2. As in the former Deterministic example, we consider the basic N model and modify the service time distribution to be Log-Normal with with coefficient of variance $CV = 1$, that is $\mu = \sigma$ as in the exponential distribution.
Namely, we take $D_{11} = LN(1,1)$, $D_{21} = LN\left(\frac{2}{3}, \frac{2}{3}\right)$ and $D_{22} = \left(\frac{1}{2}, \frac{1}{2}\right)$.

While deterministic service times in single class single pool settings $(M/D/N)$ constitutes an upper bound for the performance of the general service distribution queue $(M/G/N)$, it appears from our study (Figure 17) that this is not the case here. The optimal solution for the Log-Normal case, obtained by our algorithm and confirmed by simulation, seems to be somewhat better than the solution for the deterministic case.

Figure 15: Performance Measures for N Model Case With Deterministic Service Times

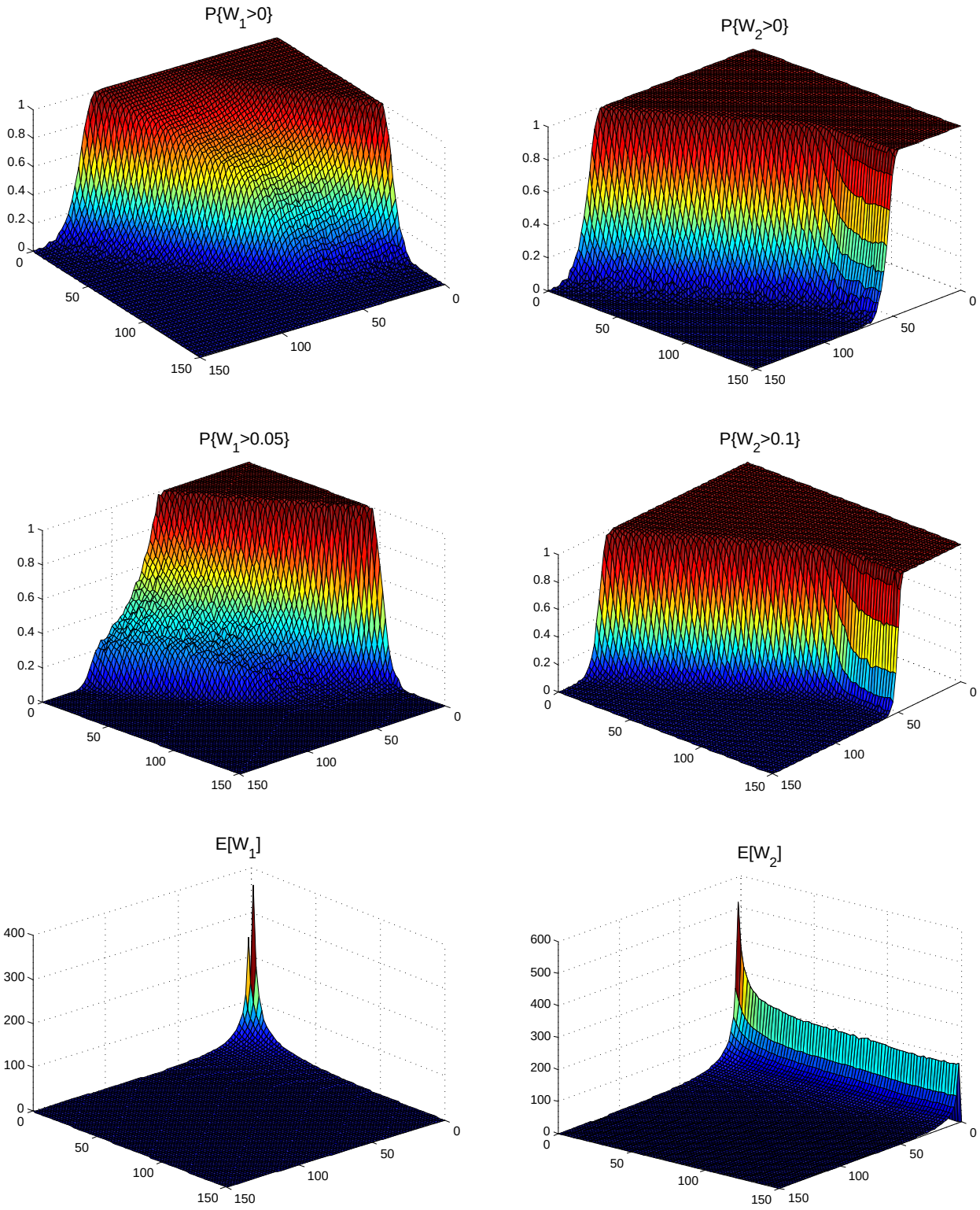


Figure 16: Performance Measures for N Model Case With Log-Normal Service Times

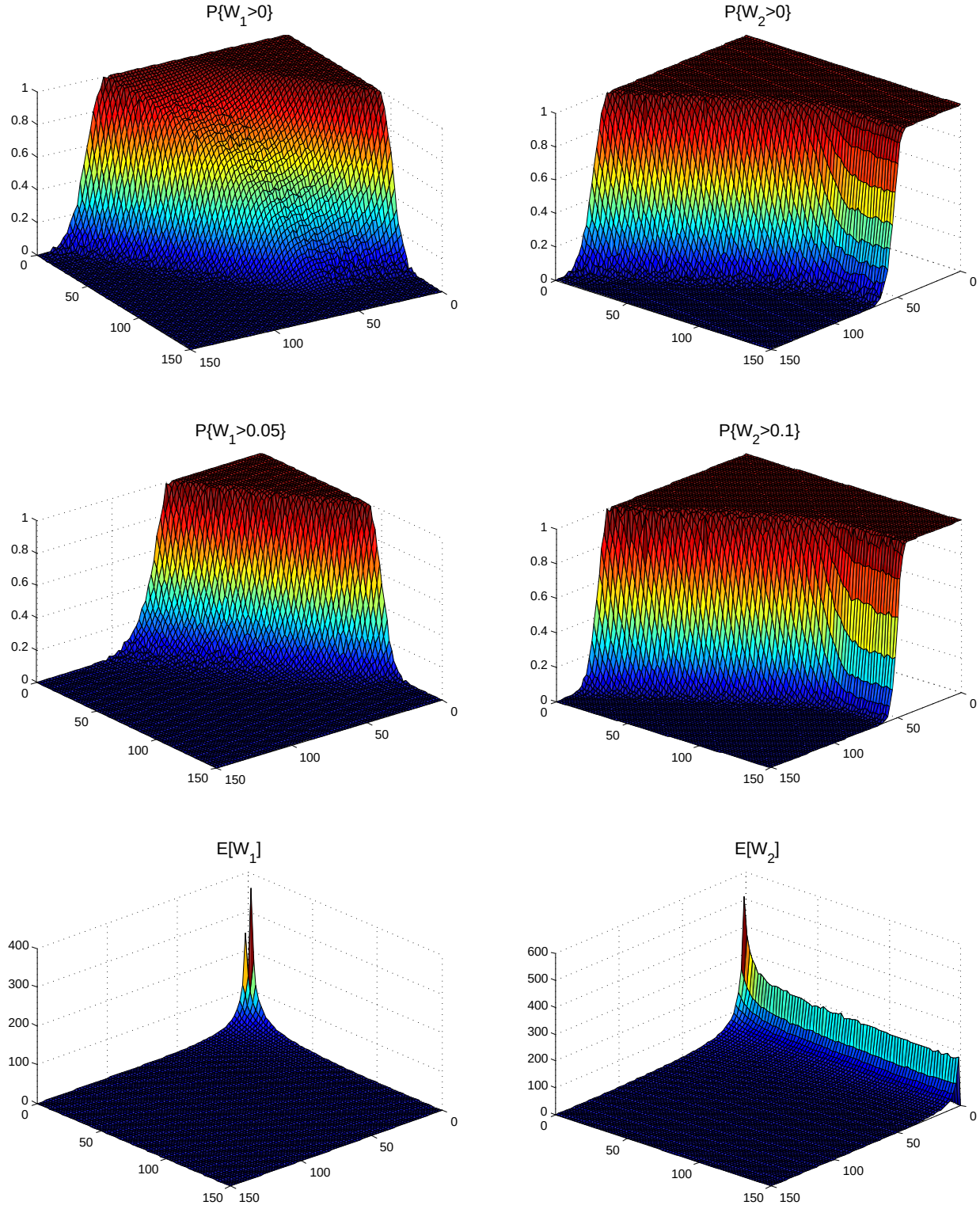


Figure 17: Feasible Region and Optimal Solution Log-Normal and Deterministic Service

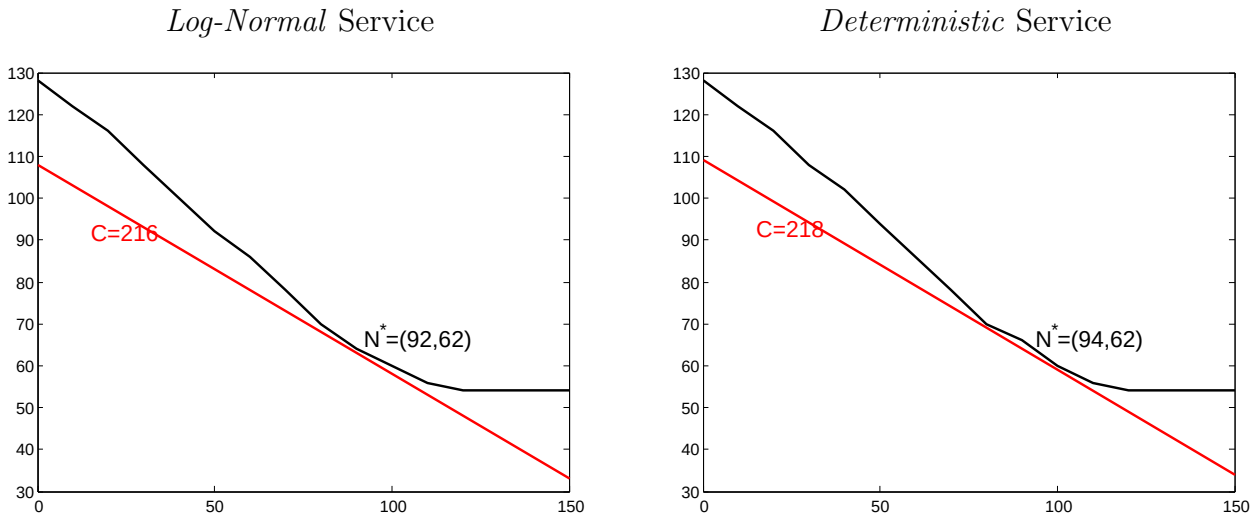
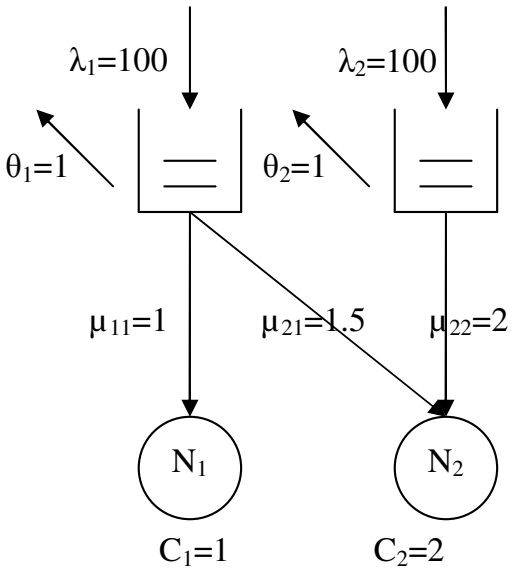


Figure 18: Schematic Representation of the N Model Settings With Abandonments



4.2.5 Adding Impatience

$$\begin{array}{ll} \min\limits_{N} & 1N_1 + 2N_2 \\ \text{s.t.} & \mathbb{P}_N\left\{ab_1\right\} \leq 0.05 \\ & \mathbb{P}_N\left\{ab_2\right\} \leq 0.1 \\ & AN \leq b \\ & N \in \mathbb{Z}_+ \end{array} . \tag{4.55}$$

Again, we try the alternative cost formulation

$$\begin{array}{ll} \min\limits_{N} & 1N_1 + 2N_2 + 300\mathbb{P}_N\left\{Ab_1\right\} + 200\mathbb{P}_N\left\{Ab_2\right\} \\ \text{s.t.} & AN \leq b \\ & N \in \mathbb{Z}_+ \end{array} . \tag{4.56}$$

Here, the optimal solution is $N^* = (102, 56)$.

In addition, we demonstrate in Figure 22 the convergence of the SA algorithm to the optimal solution. At the top plot we display the convergence of each component of the approximation $\bar{x}_j$, and at the bottom plot the path of the iterative process x_j along with the instantaneous approximation $\bar{x}_j$ are displayed on the solution space. The algorithm started off at point $\bar{x}_1 = x_1 = (140, 180)$ and ended at $\bar{x}_{3000} = (98, 58)$.

4.3 M Model

We expand the model from 4.2 by adding another pool of servers, this time the servers in this pool are serving only customer of $class_2$. The servers of this pool has the same cost as pool 1 but they serve $class_2$ customers with rate $\mu_{32} = 2.5$. The stability conditions in this case are

$$\begin{array}{l} N_1 + 1.5N_2 \geq 100 \\ 2N_2 + 2.5N_3 \geq 100 \\ 0.66N_1 + 1.25N_2 + N_3 \geq 116 \end{array} \tag{4.57}$$

The problem formulation is given by

$$\begin{array}{ll} \min\limits_{N} & 1N_1 + 2N_2 + 2N_3 \\ \text{s.t.} & \mathbb{P}_N\left\{W_1 > 0\right\} \leq 0.2 \\ & \mathbb{P}_N\left\{W_2 > 0\right\} \leq 0.5 \\ & AN \leq b \\ & N \in \mathbb{Z}_+ \end{array} , \tag{4.58}$$

Figure 19: Performance Measures for N Model Case With Abandonments

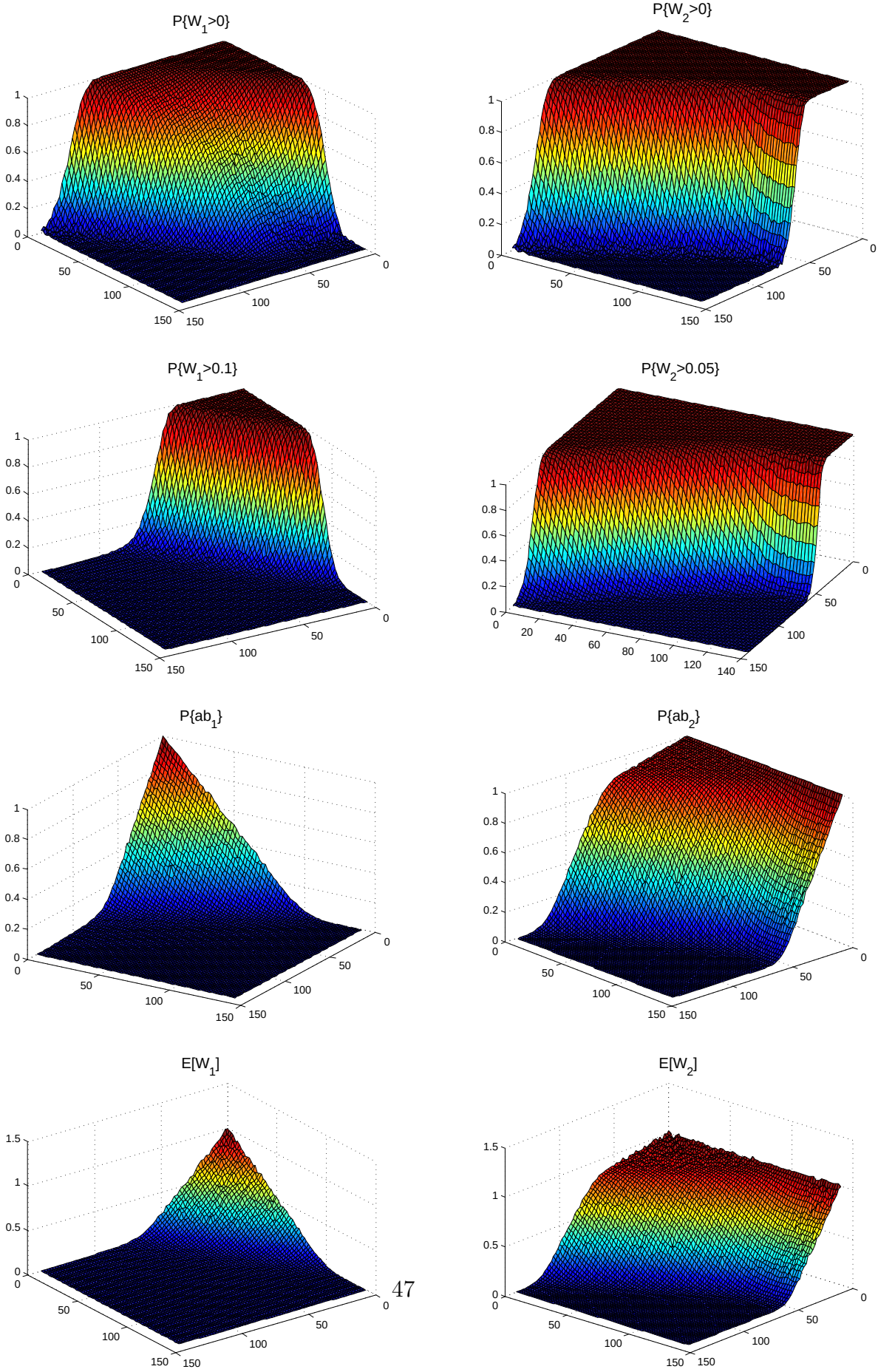


Figure 20: Feasible Region and Optimal Solution for Target Abandonment Probability

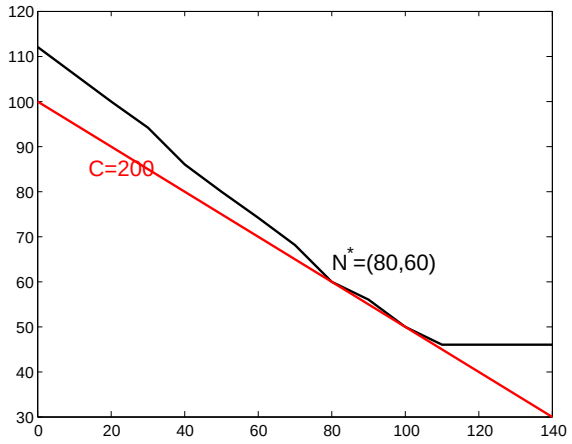


Figure 21: Cost Function for N Model Optimization Formulation With Abandonments

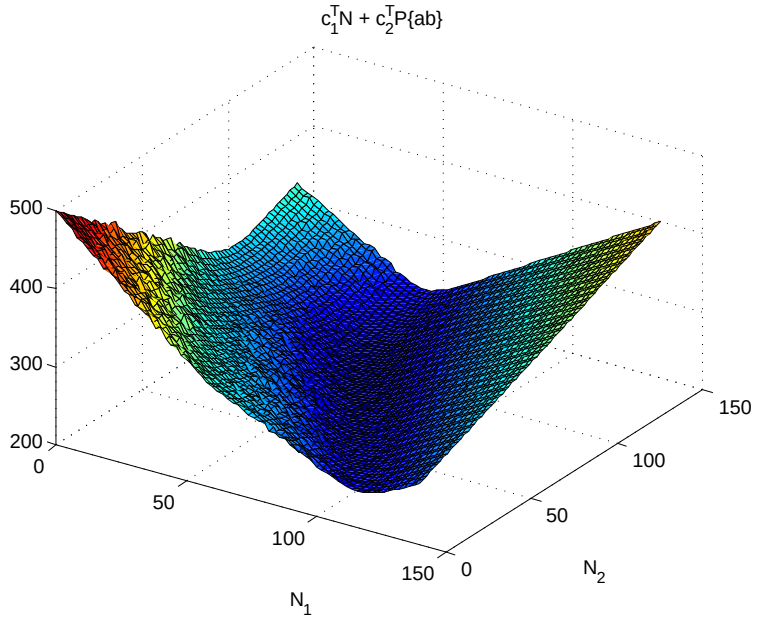
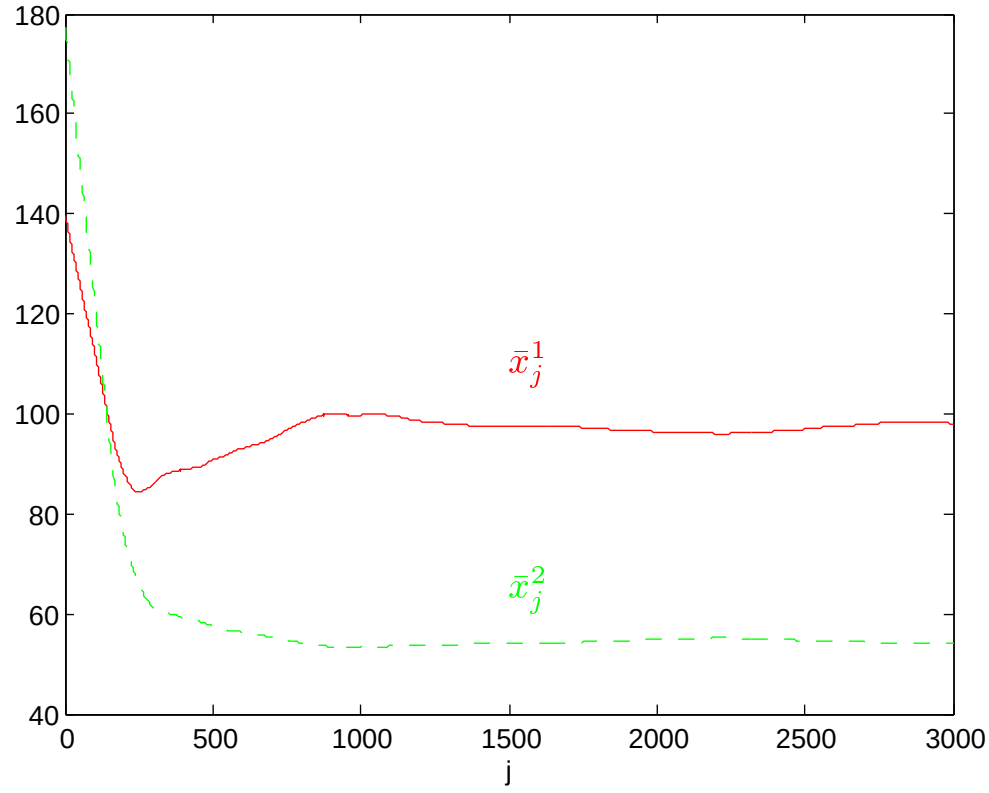


Figure 22: Convergence To The Optimal Solution



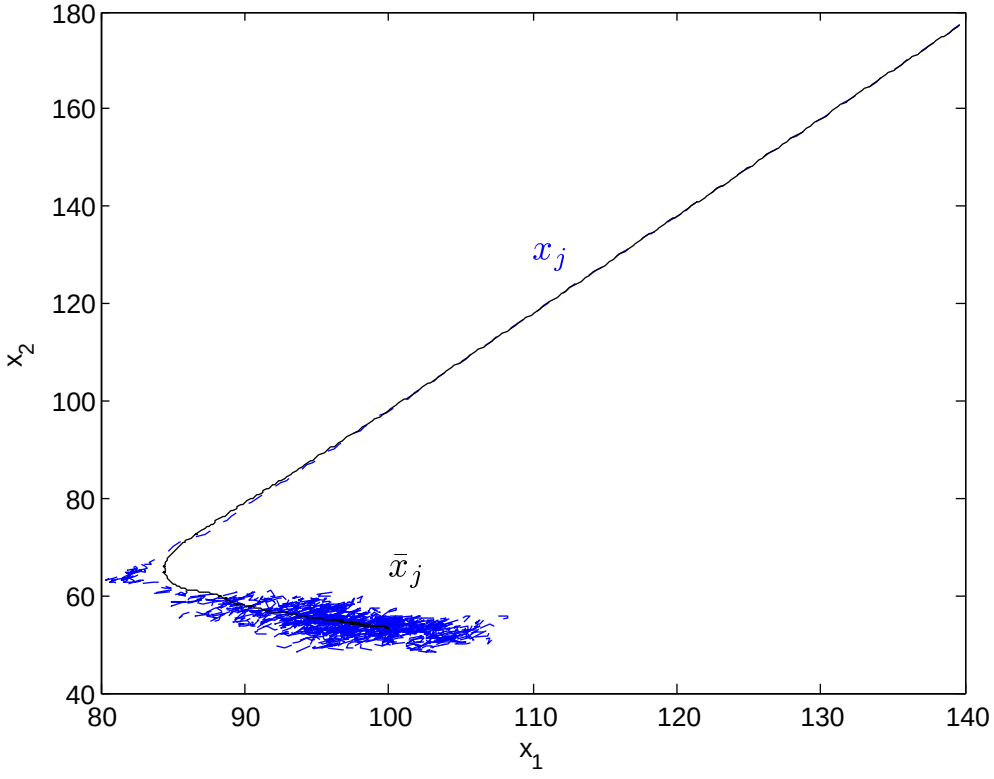
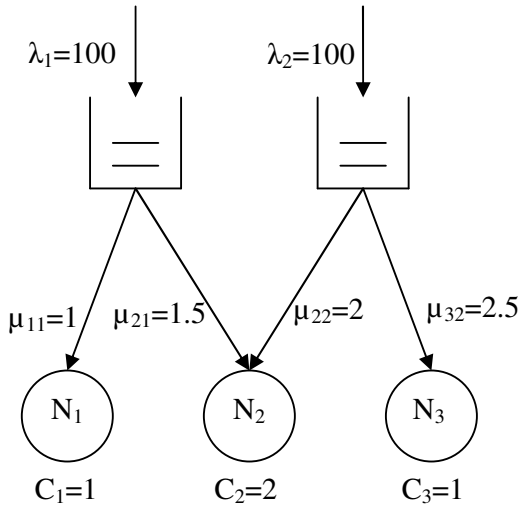
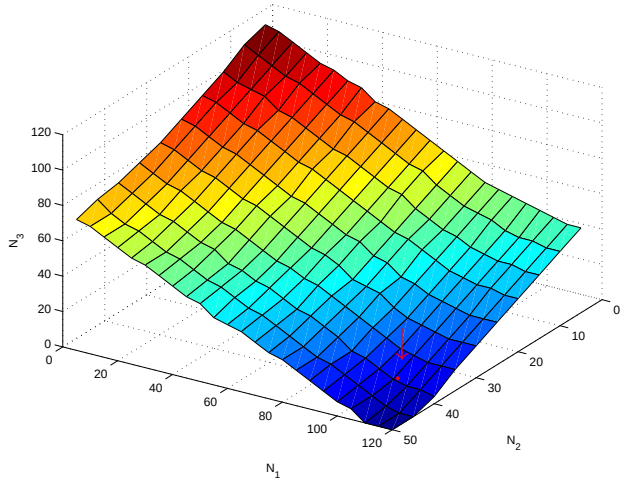


Figure 23: Schematic Representation of The M Model Settings



where $A = \begin{bmatrix} -1 & -1.5 & 0 \\ 0 & -2 & -2.5 \\ -\frac{2}{3} & -1.25 & -1 \end{bmatrix}$ and $b = \begin{bmatrix} -100 \\ -100 \\ -117 \end{bmatrix}$. The feasible region appears in 24 as the area above the displayed surface. The optimal solution $N^* = (105, 37, 9)$ is marked as a red point pointed by an arrow. In this case, the optimal solution was obtained exactly.

Figure 24: Feasible Region and Optimal Solution



4.4 Time-varying Model

We consider here models in which arrival rates vary with time, and service levels are measured in various time bases. (e.g. the delay probability may be maintained on an hourly basis, daily or even weekly). Consequently, staffing levels are also allowed to change with time in order to meet with the timely constraints or align with the timely cost optimization.

There are additional operational complexities arising from the extension to a time-varying environment. First, a policy should be pre-determined for what to do in case of a decrease in staffing level. Two main alternatives are the *preemption discipline* and the *exhaustive discipline*. According to the preemption discipline, whenever there is a decrease in staffing level, the last-to-enter-service customers are removed from service and returned to the front of their queue; the number of removed customers equals the decrease in the staffing level. This situation seems to be unreasonable for most service systems. The Call Center reality is probably much closer to the exhaustive discipline, stipulating that servers can be released only after they complete their present service.

In the most general case, we may want to optimize a system on a given time horizon, which may differ from Service Level time basis and from the planning period[5]. For instance, we may wish to optimize the staffing levels of a whole week, satisfying daily service level constraints, and considering hourly planning period. We are dealing with this situation as follows. We are first dividing the entire time horizon into intervals of the same length of the Service Level time basis. Since Service level must be maintained for each one of those intervals, we are able to treat the global optimization problem as several smaller independent problems that are solved sequentially, interval by interval. The state at the end of each interval is taken as the state at the beginning of the consecutive interval.

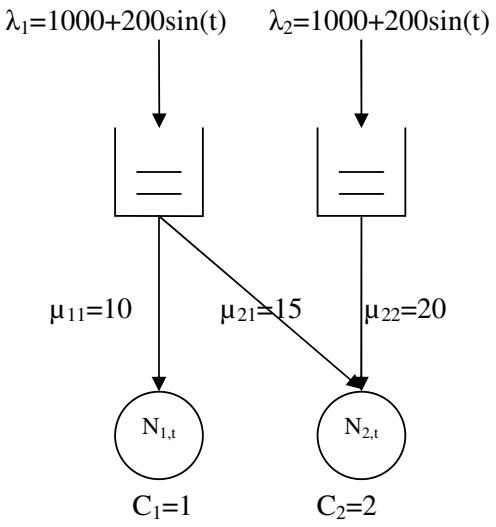
For the purpose of solving each smaller problem we must generalize our algorithm. Similar to what is carried out in [3], we differentiate the Service Level with respect to the staffing level in each pool, on each planning period. We will thus have to make the assumption that the service level functions are convex in the $J \times T$-dimensional staffing vector, where J is the number of pools and T is the number of planning periods. To facilitate, following the example from above, we will divide the week into seven days, and solve each day as a separate problem starting from the first day. Within the day we will estimate the gradient

[5] we refer to planning period as the shortest amount of time over which staffing must remain constant

in each iteration of our algorithm by increasing the staffing levels of each pool in each hour. Once we complete the solution of the first day, we will start with the second day, having the state of end of the first day to be the initial state on the second day.

Let us consider an N-model with time varying arrival rates. $class_1$, as well as $class_2$ customers, arrive according to an inhomogeneous Poisson process with arrival rate $\lambda_1(t) = \lambda_2(t) = 1000 + 200\sin(t), t \in [0, 10]$. Servers from $pool_1$ serve $class_1$ at rate $\mu_{11} = 10$ and $class_2$ with rate $\mu_{12} = 15$. $pool_2$ servers serve only $class_2$ with rate $\mu_{22} = 20$.

Figure 25: Schematic Representation of The Time-Varying N-model Settings



In addition, delay probability constraints should be maintained on an hourly basis and our planning period is also an hour, i.e. the staffing levels can vary every hour but not within the hour. Formally, we consider the following problem

$$\begin{array}{lll} \min\limits_{N} & \sum_{t=1}^{10} 1N_{1,t} + \sum_{t=1}^{10} 2N_{2,t} & \\ \text{s.t.} & \mathbb{P}_N\{W_{1,t} > 0\} \leq 0.1 & \forall t = 1, \ldots, 10 \\ & \mathbb{P}_N\{W_{2,t} > 0\} \leq 0.5 & \forall t = 1, \ldots, 10 \\ & N \in \mathbb{Z}_+^2 0 & \end{array}, \tag{4.59}$$

where $W_{i,t}$ is the waiting time of $Class_i$ customer arriving at interval $[t-1, t]$, $N_{j,t}$ is the staffing levels in Pool j at time interval $[t-1, t]$, and $N_t = (N_{1,t}, \ldots, N_{J,t})$.

We solve this problem by dividing the 10 hours to 10 problems of 1 hour. We start in the first hour, letting the final state given the optimal solution be the initial state for next hour and so on.

The results of the algorithm and the service levels are displayed in Figures 26, 27 and 28. In Figure 26 we can see the arrival rates and staffing levels that were obtained by the algorithm. Note that the arrival rates do not follow exactly the sinusoidal function. This is because the values are averaged for each time interval of length 1. In Figure 27 we can see that the target delay constraints was satisfied at all times. The moderate fluctuations in service level and staffing levels are the consequence of having several optimal cost solution, each of which incurs a slightly different service level. In case there are several optimal solutions, we do not have control over which solution the algorithm will converge at. In Figure 28 two types of graphs are displayed. The first two graphs display the occupation profile of each servers pool, i.e. the proportions of the time spent on serving each class of customers, and being idle out of the total available time of all servers on that pool and time interval. The values are stacked so that their total sums up to 1 (100 %). Interestingly, it appears that these proportions remain rather stable throughout all time intervals. The bottom graphs display for each the average number of customers in each pool and in queue for any time interval. The values are stacked so that the total value represents the average number of customers in the system.

4.5 Realistic Example

In this example, we take data of a random day from a real medium-size Call Center, providing different types of banking services. We focus on two types of calls: calls coming from Business customers and Quick Request calls.

The arrival rates of the incoming calls is depicted in Figure 29.

Furthermore, we try to fit a distribution for the service times. We assume that the service times are dependent of the class, but do not depend on the server taking the call, nor on the time of day.

As shown in Figure 30, the Log-Normal distribution seems to be a good fit for the service times of both classes. We thus take the service distribution of the Business class to be $LN\,(0.063, 0.058)$ and the service distribution of the Quick Request class is $LN\,(0.065, 0.073)$.

Figure 26: Arrival Rate and Staffing Function for The Time-Varying Example

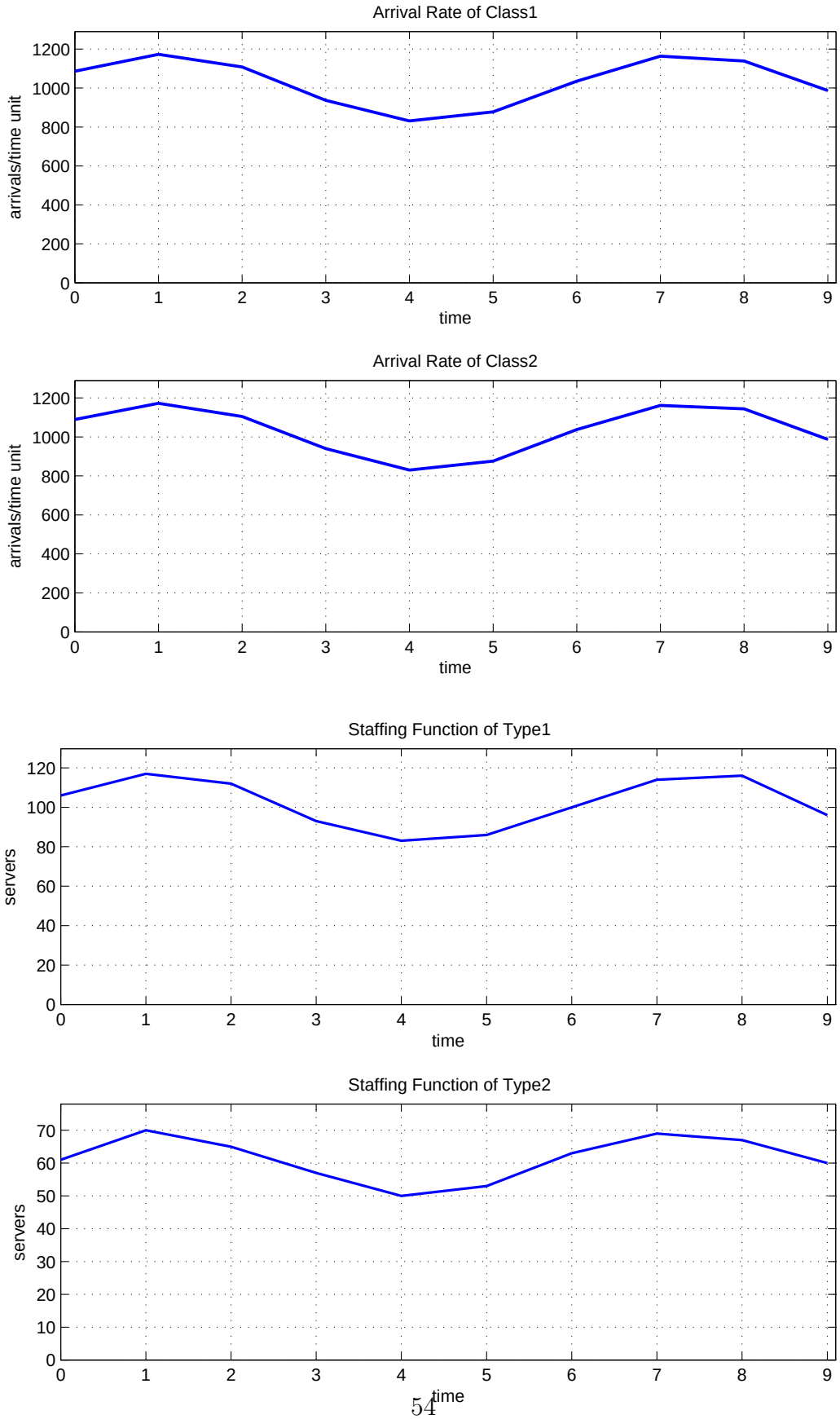


Figure 27: Delay Probability and Tail Probability for The Time-Varying Example

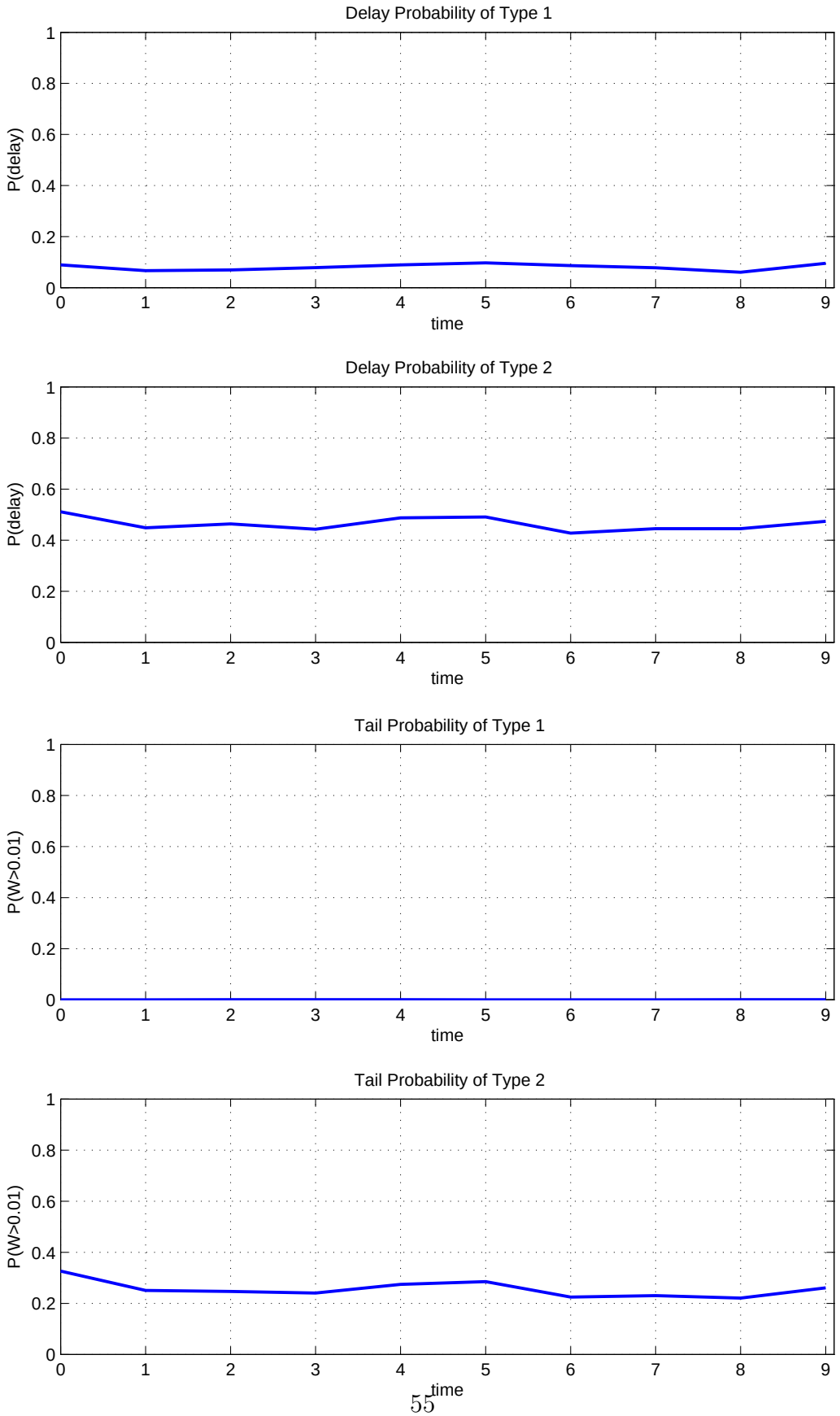


Figure 28: Utilization Profile and Customer Distribution for The Time-Varying Example

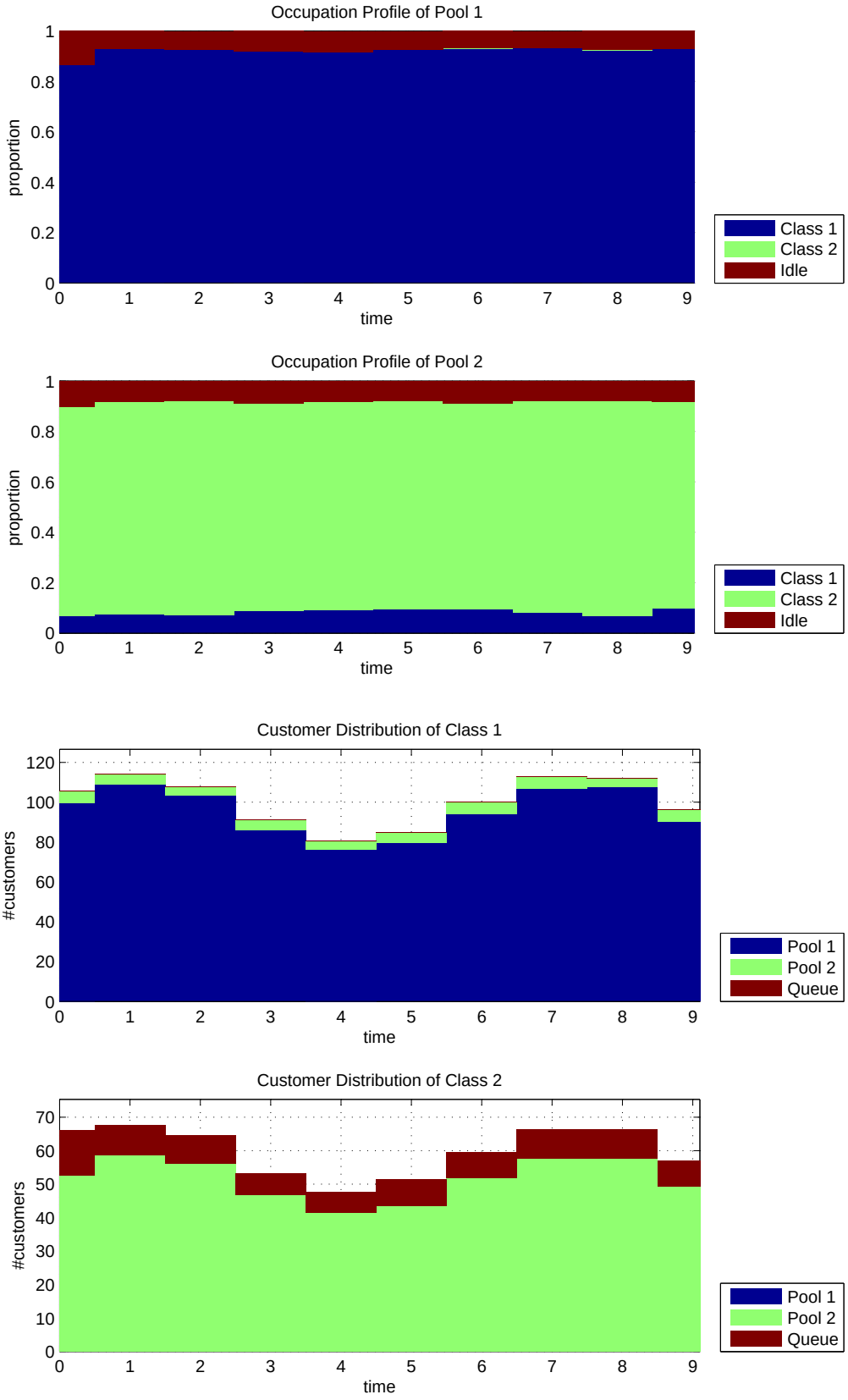
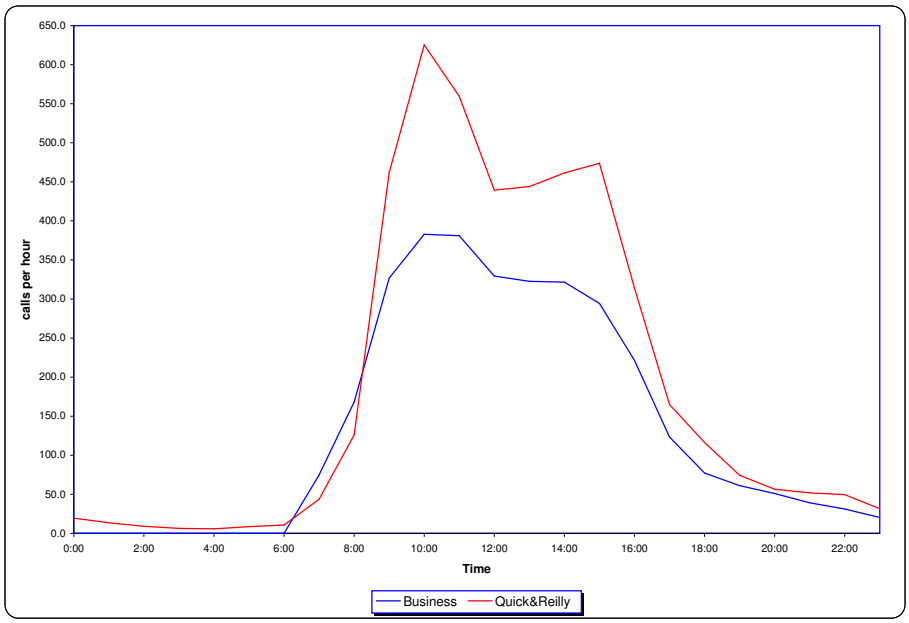


Figure 29: Call Volumes to a Medium-Size Call-Center



We also use Survival analysis to fit a distribution for the patience of customers, i.e. the time they are willing to wait for service before they abandon the call. Our analysis shows that the patience of both classes fits to the Exponential Distribution rather well (see Figure 31), and that there is a significant difference in the mean time. Business customers are willing to wait on average 7.35 minutes, while Quick Request customers are willing to wait as much as 19.35 minutes on average.

There are two types of servers in this system. One type is catering to both classes, while the other is dedicated only to Business customers, and their cost are similar.

For start, we want to find the minimal cost staffing levels for which not more than 10 percents of Business customer and not more than 50 percents of Quick Request customers will have to be delayed on any hour. More formally we would like to solve

$$\begin{array}{ll} \min\limits_{N} & \sum_{t=1}^{24} N_{1,t} + \sum_{t=1}^{24} N_{2,t} \\ \text{s.t.} & \mathbb{P}_N\left\{W_{1,t} > 0\right\} \leq 0.1 \qquad \forall t = 1, \ldots, 24 \\ & \mathbb{P}_N\left\{W_{2,t} > 0\right\} \leq 0.5 \qquad \forall t = 1, \ldots, 24 \\ & N \in \mathbb{Z}_+^{48} \end{array}, \tag{4.60}$$

We run our algorithm and obtained the solution as presented in 32 with the total labor of 575 hours. The target was obtained at all hours (Figure 33). Moreover, it appears that the average waiting time is negligible (few seconds) throughout the entire day.

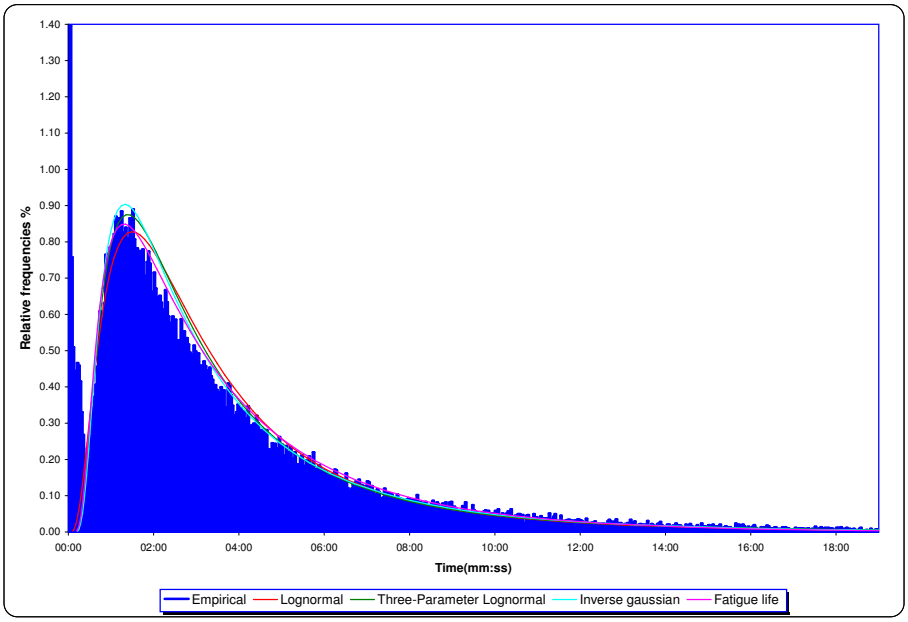
We now try to run our algorithm with daily service level constraints, that is, the delay probability targets should be maintained on the overall arrivals.

$$\begin{array}{ll} \min\limits_{N} & \sum_{t=1}^{24} N_{1,t} + \sum_{t=1}^{24} N_{2,t} \\ \text{s.t.} & \mathbb{P}_N\left\{W_1 > 0\right\} \leq 0.1 \\ & \mathbb{P}_N\left\{W_2 > 0\right\} \leq 0.5 \\ & N \in \mathbb{Z}_+^{48} \end{array}, \tag{4.61}$$

In the solution obtained by the algorithm, the number of working hours was reduced by 65 hours (11%).

The daily probability of delay was satisfied ($\mathbb{P}\left\{W_1\right\} = 0.19$ and $\mathbb{P}\left\{W_2\right\} = 0.32$). However, as shown in Figure 36, while the service levels were very good during the rush hour (8:00 am until 17:00 pm), essentially all the customers that arrived outside of these hours

Figure 30: Service Distributions



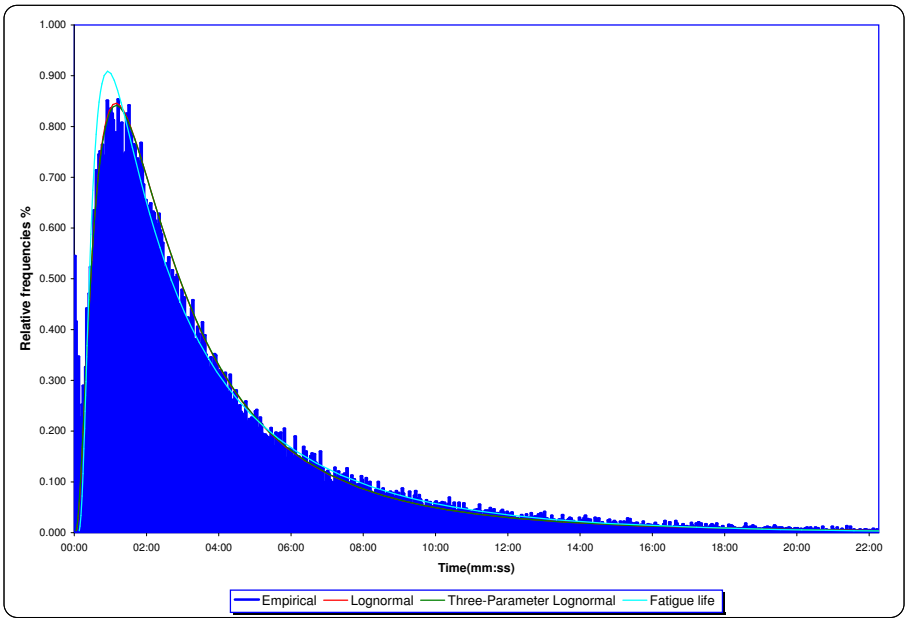
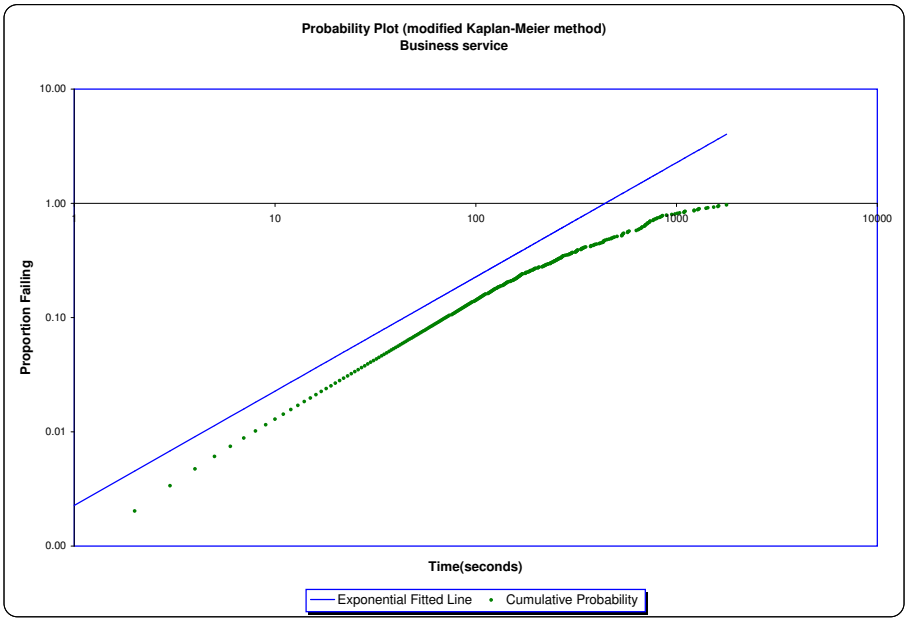


Figure 31: Patience Distributions Analysis



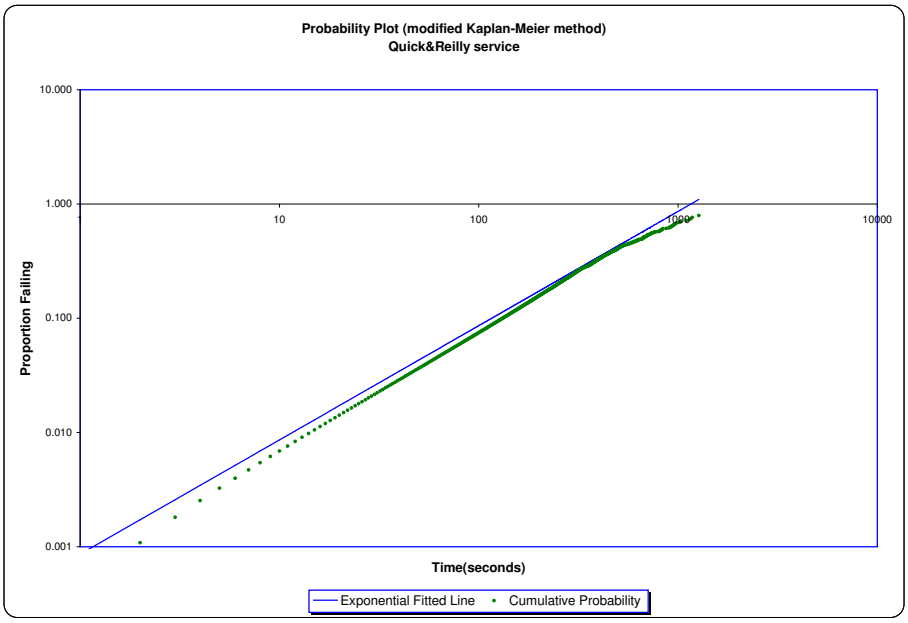


Figure 32: Staffing Levels for Satisfying Hourly Delay Probability

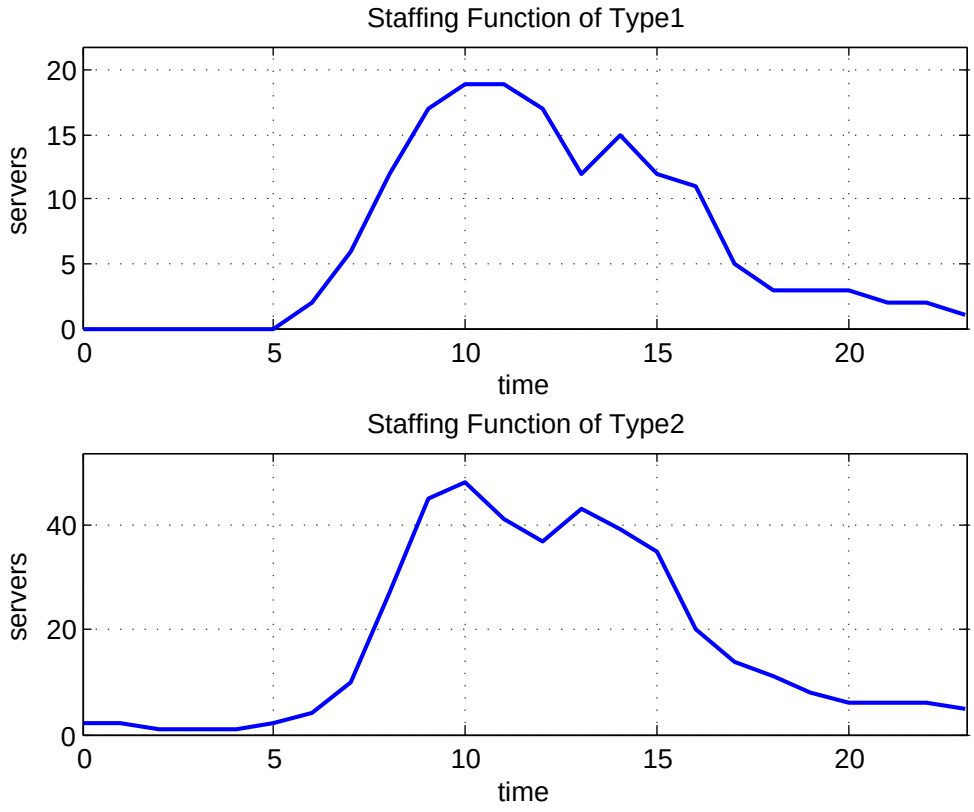


Figure 33: Delay Probability

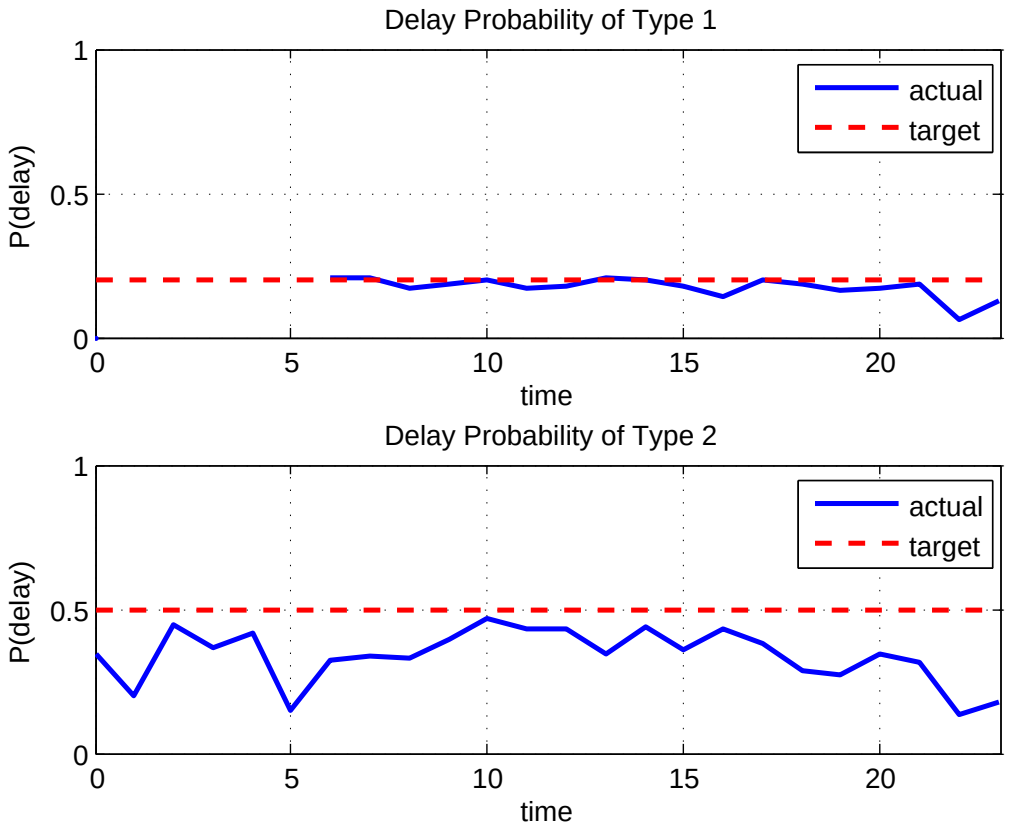


Figure 34: Average Wait

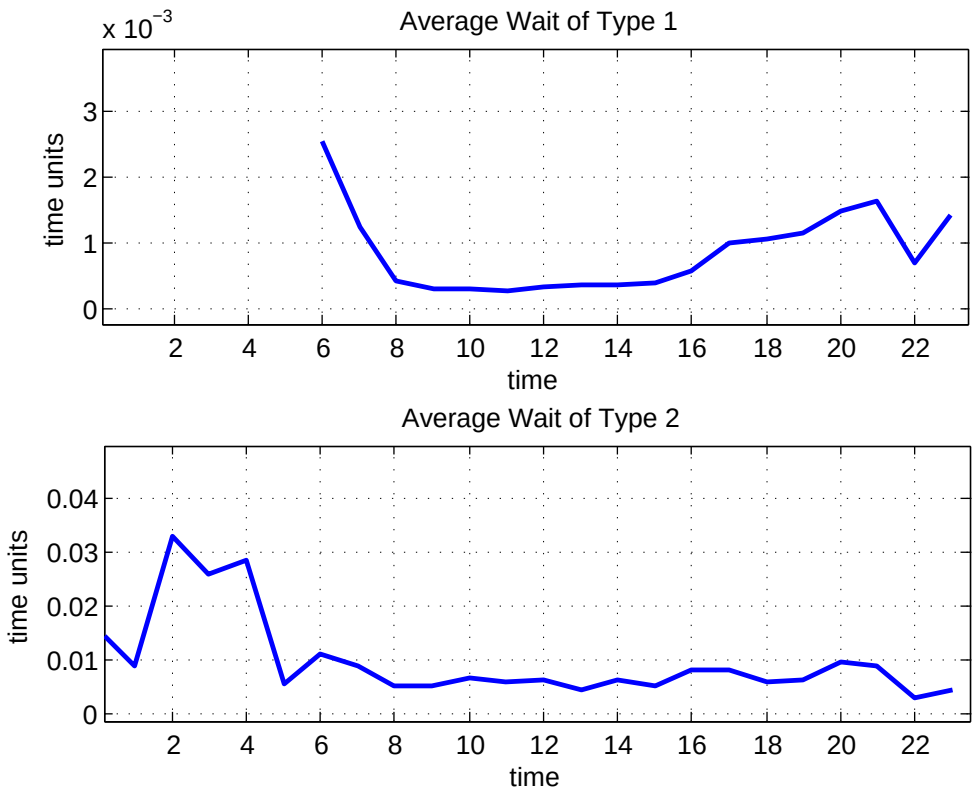
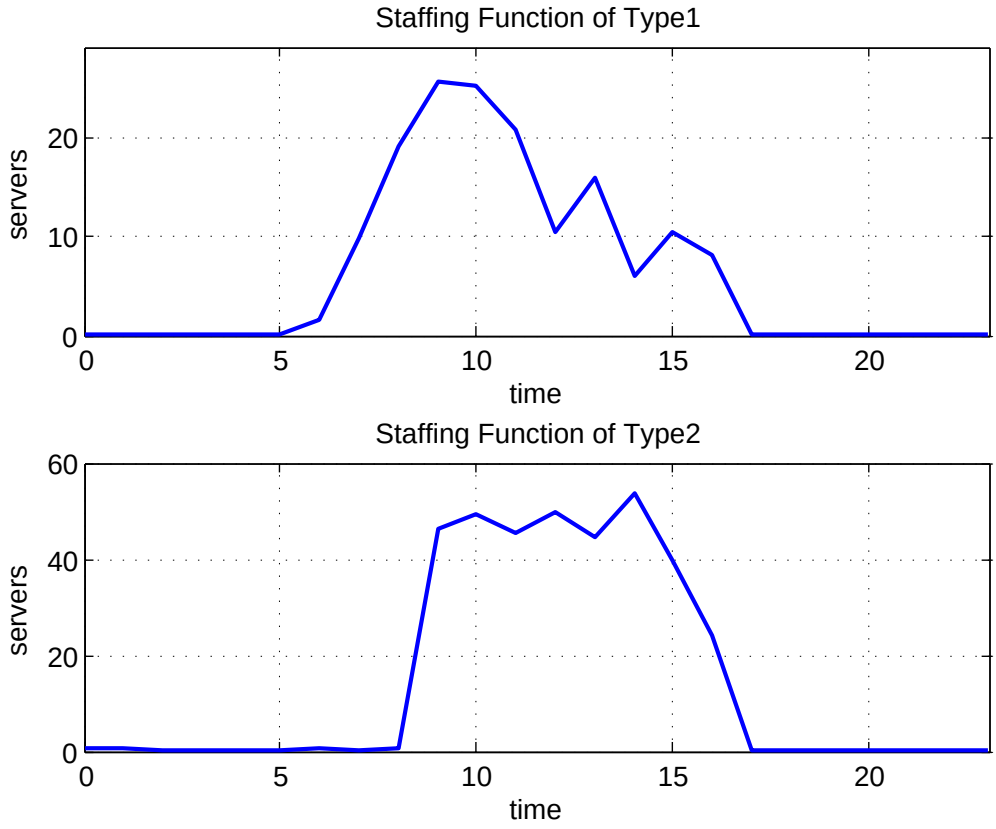


Figure 35: Staffing Levels for Satisfying Hourly Delay Probability



were delayed and encounter a bad service experience (average wait is several minutes). Moreover, the two solution are not totaly comparable in term of service level, since the status at the end of the horizon is different. While there are very few customers on the queues during the entire horizon when service level is satisfied on each hour, a sizable queue was built up at the end of the horizon when the service level was maintained on a daily basis. Consequently, more agents will have to be assigned on the following day to compensate.

Smaller difference in total required staffing hours was detected, when we tried to satisfy constraints on the probability to abandon both hourly and daily. We demanded that not more than 10% of the Business customers will abandon, and not more than 20% of the Quick Requests customers.

The total amount of staffing hours was 463 for the hourly service level, and 432 for the daily service level (approximately 6.7% reduction). Again, in the case of daily constraints,

Figure 36: Delay Probability

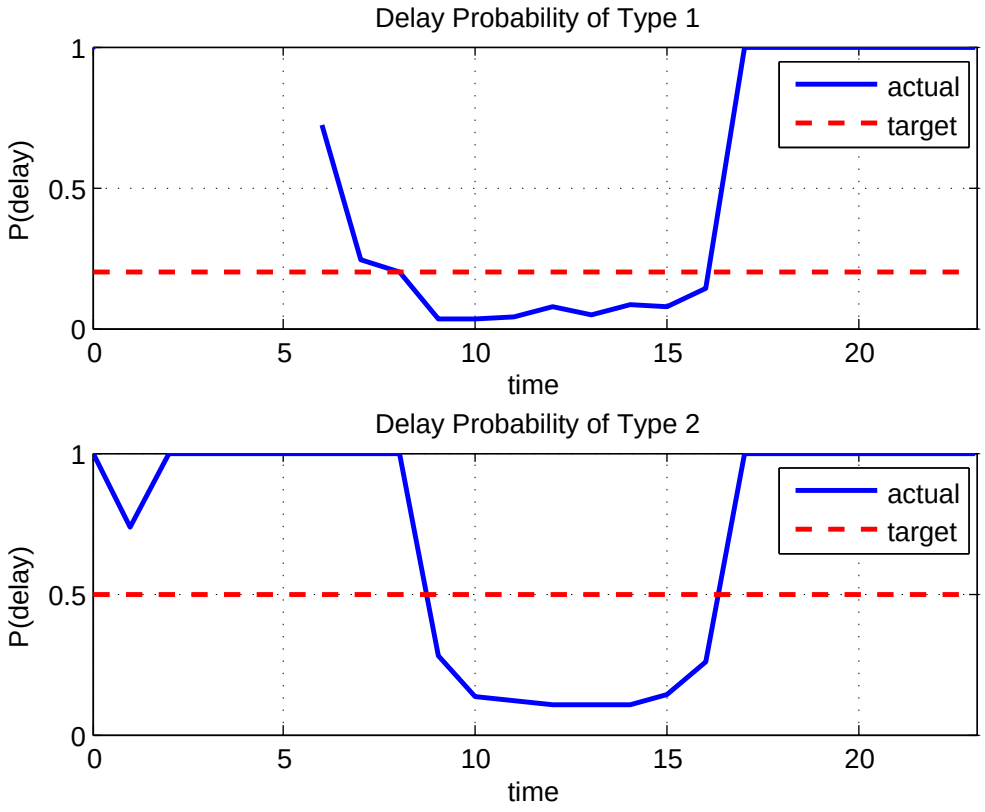


Figure 37: Average Wait

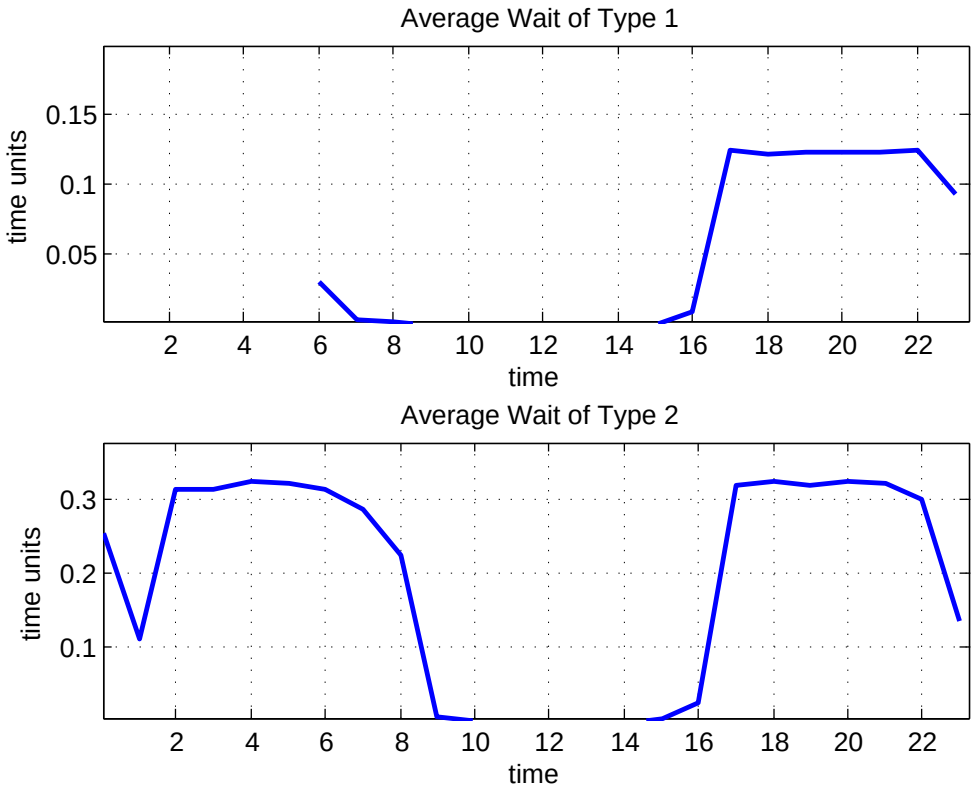
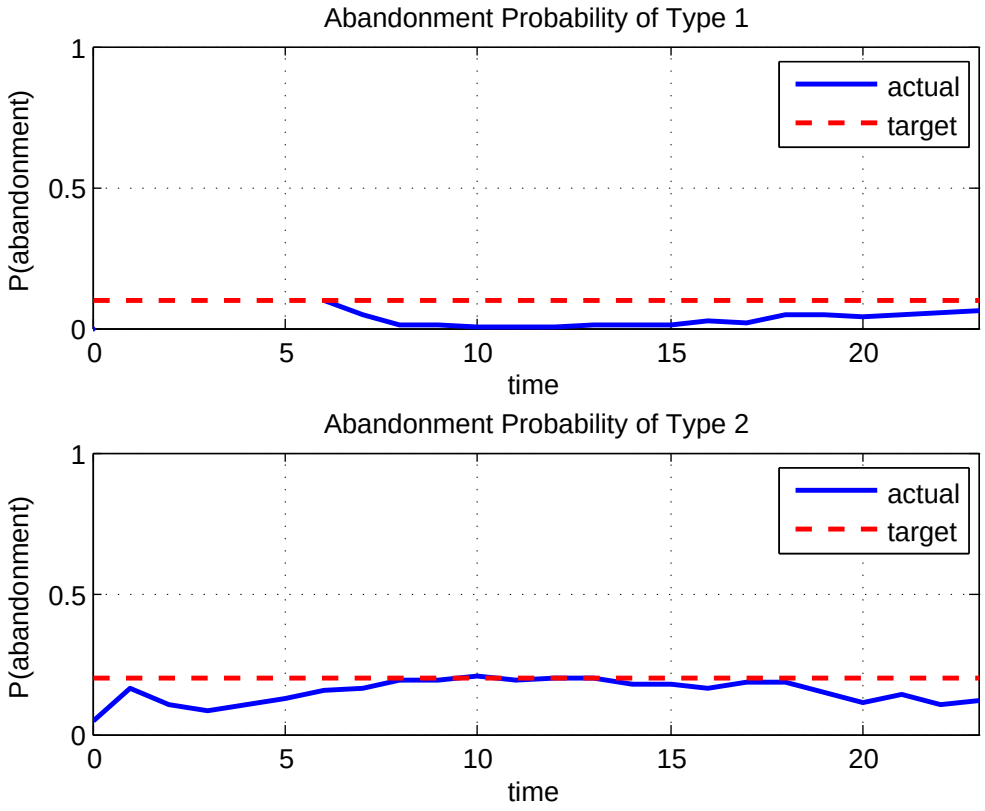


Figure 38: Abandonment Probability for Hourly Service Levels



the probability to abandon was below the target during the rush hours in which the critical mass of customers arrives, and significantly worse on the rest of the hours (Figures 38,39).

In another experiment that we carried out, we associated a cost to the waiting time of customers and tried to find staffing levels that minimize the total staffing cost and waiting time cost. We set the cost of one hour of wait to be 0.2 cost units for the Business customers, and only 0.1 for the Quick Requests customers. Much like in the constraints satisfaction mode, the cost optimization mode can also be applied with different service level time base. However, there was no significant difference in the solution when we tried different time bases.

Figure 39: Abandonment Probability for Daily Service Levels

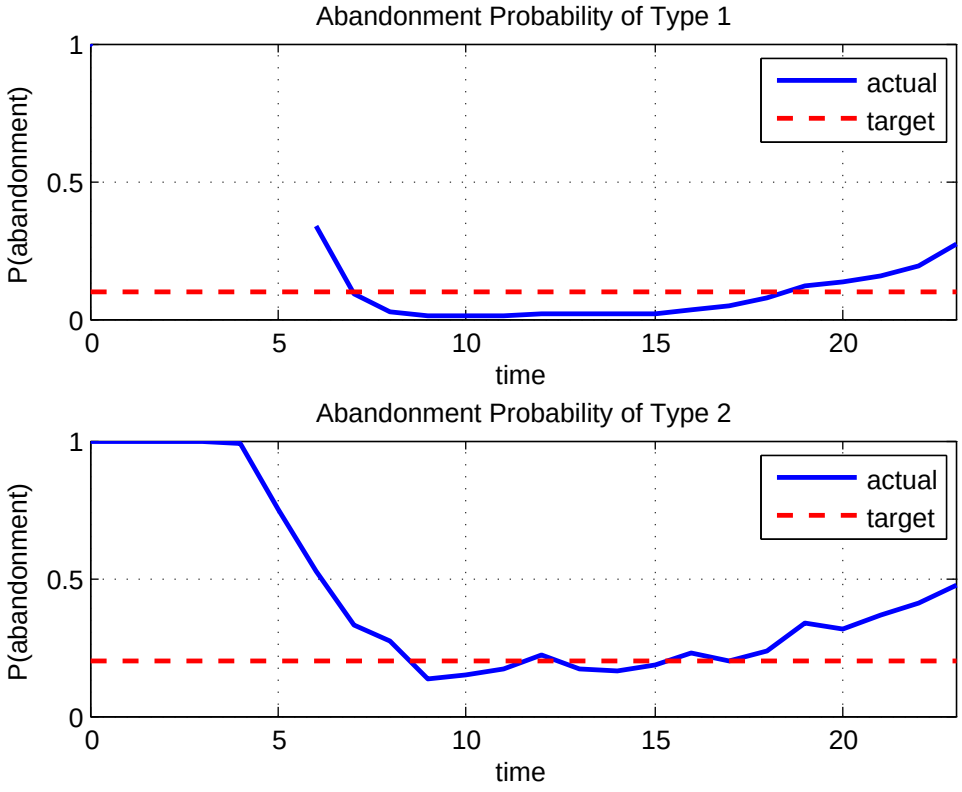
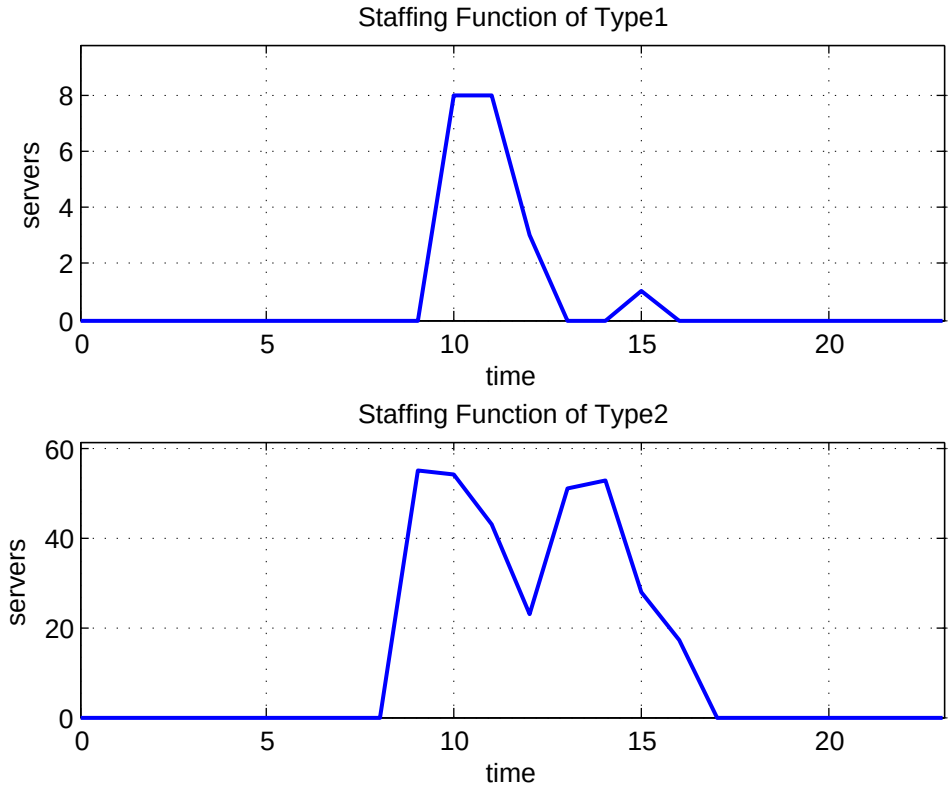


Figure 40: Staffing Levels for The Waiting Cost Optimization



5 Staffing Optimizer Tool

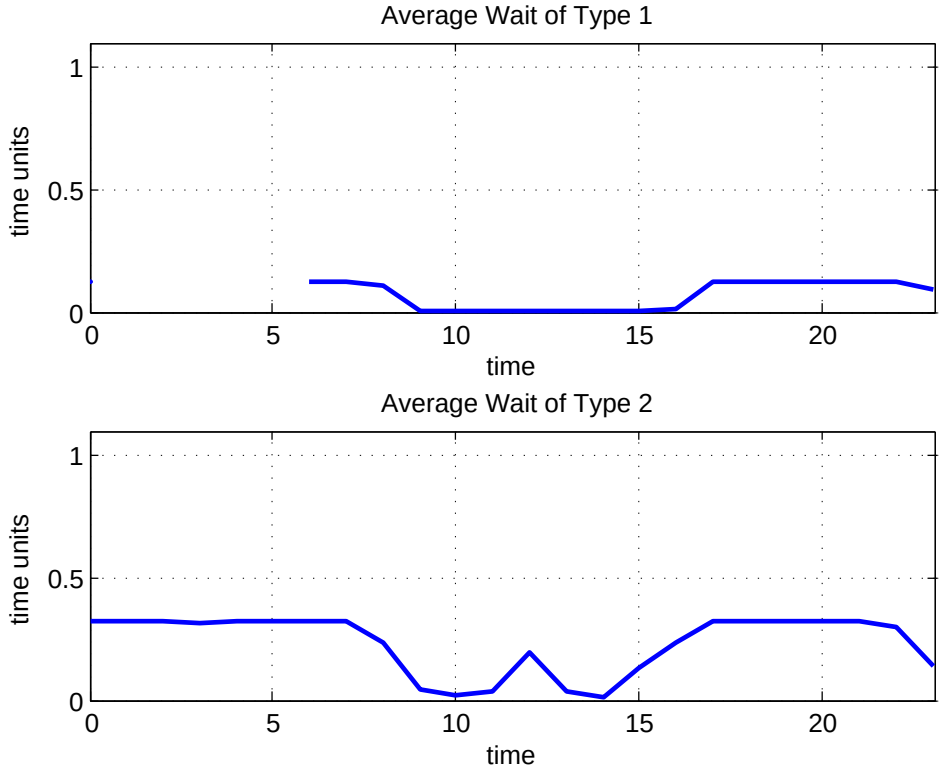
In this section we give general outlines of the simulation we developed for SBR systems with time-varying parameters. Furthermore, we describe a MATLAB interface that is used for creation of simulation models, optimizing staffing, running simulation and viewing their results.

5.1 Simulation Description

We give here the general design of our simulation including a short description of the main classes and their associations. Our simulation is written in C++. In addition to the simulation library, we created a dynamic link library (dll) that provides simple API for creation of simulation models and running them according to user parameters. This library is then linked to our MATLAB interface code.

The main classes:

Figure 41: Average Wait for The Waiting Cost Optimization



CustomerClass This class implements a type of customers. It includes data about the arrival rate function, the distribution of patience associated with customers from this class, thresholds for waiting times and sojourn times that are used for statistics gathering. The main functionality of this class is generating Customer objects and queueing them.

ServerPool This class implements a pool of servers. It holds the staffing function of this pool and the service discipline. Service discipline dictates what happens when staffing level is reduced and can be either *exhaustive* or *preemptive*. In case it is *exhaustive*, server that has to leave will first finish the service and only then leave. In case of preemption discipline, the server will leave immediately and the customer will be returned to the head of the queue.

Connection This class defines the service information for a certain customer class and a pool of servers (if a certain pool cannot serve a certain class of customers, there will not be a connection defined for them). The information includes the service distribution, and the control parameters.

ServiceSystem This class implements a skills-based-routing system that is composed of one or more customer classes, one or more pools of servers and the topology defined by the set of connections between them. It contains the data of control schema, both arrival control and Idleness control, the grid according to which statistics is gathered. the main functionality of this class is running the simulation according to user parameters: start time, end time, and the number of replications. The simulation is event driven and based on heap of events processed by event handler.

Event Events are the building blocks of the simulation. There are five types of events:

- *Customer Arrival* - Customer enters the system. He is routed to a pool according to the routing scheme or queued if there are no idle servers capable of serving him.
- *Customer Release* - Customer is released from service after it completed. The newly idle server is taking another customer to service according to the routing scheme or remains idle if there are no customer that he can serve.
- *Customer Abandon* - Customer abandons the system and removed from queue.
- *Change Shift* - number of servers and customers are handled according to the service discipline.
- *Report Time* - Statistics regarding the number of customers of each class is gathered.

Events are entered into a heap. Events handling process invoke the creation of other events.

Customer Customer objects are generated by the *CustomerClass* objects. They are used to keep track of time stamps along the service process and report statistics.

Control Control implements the routing scheme according to which customers are routed to servers upon their arrival (**Arrival Control**), and idle servers choose a customer to serve (**Idleness Control**). There are several control schemes implemented in the code.

1. **Arrival Control**

   We say that a pool is a candidate for a customer at a given time if servers in

this pool are able (there is a connection between the pool and class) to serve the customer, and there is at least one available server in this pool.

- Random (RAND) - a customer will be routed a randomly (with equal probability) to any of the candidate pools at the time of arrival.
- Static Priority (SP) - a customer will be routed the to a candidate pool with the highest priority according to a predefined class-dependent priority list.
- Threshold Priority (TP) - A customer will be routed to the highest priority candidate pool that has at least K idle servers. The threshold K is defined for each pool and class.
- Preemptive Static Priority (PSP) - A customer will be routed to the highest priority candidate pool that is either available or has at least one server that serves a customer with lower priority. In the latter case , the lower priority customer will be removed from service back to the head of his queue for the benefit of the coming customer.
- Fastest Server (FS) - A customer will be routed to candidate pool that serves him with the lowest expected service time.
- Fixed Queue Ratio (FQR) - A customer will be routed to a candidate pool j with the highest value of $\frac{I_j}{\sum I_j} - p_j$, where I_j is the number of idle servers in pool j and p_j is a predefined parameter.
- Mixed Priority (MP) - Each class can be either preemptive class or not. A preemptive class customer will be routed according to the PSP scheme, else he will be routed according to SP scheme. In case a customer is removed from service he will be inserted to the end of queue of all ejected customers.

2. **Idleness Control**

- Threshold Priority (TP) - A newly become idle server will take the head-of-queue customer from a candidate class i and there are at least K_i available servers on that pool.
- Static Priority (SP) - A newly become idle server will take the head-of-queue customer with the highest priority
- Most Delayed (MD) - A newly become idle server will take the head-of-queue customer with the longest wait.

- Fixed Queue Ratio (FQR) - A newly become idle server will take the head-of-queue customer for the queue that maximize the value of the index $\frac{Q_i}{\sum Q_i} - p_i$, where Q_i is the number in queue of class i and p_i is a predefined parameter.
- Fixed Wait Ratio (FWR) - A newly become idle server will take the head-of-queue customer for the queue that maximize the value of the index $\frac{W_i}{\tau_i} - p_i$, where W_i is the waiting time of the head-of-queue customer of class i, τ_i is a predefined threshold parameter and p_i is a predefined ratio parameter.
- Mixed Priority (MP) - A newly become idle server will take the head-of-queue customer from the preemptive classes. If all preemptive classes queues are empty he will take a the customer from the head of the queue of all ejected customers. If this queue is empty he will take a customer from the non-preemptive class queue with the highest priority.
- Servient (SER) - A newly become idle server will take a customer that he can serve from any pool with higher priority. If there is no such customer he will take a customer from one of the queues according to SP scheme.

Statistics This class stores the relevant data for all pools and classes and time intervals in arrays. The types of data that are stored include the following:

- Arrivals - stores for each Customer Class an array of how many customers of that class arrived in each interval.
- Delays - stores for each Customer Class an array of how many customers were delayed before admitted to service in each interval.
- Abandonments - stores for each Customer Class an array of how many customers that arrived in a specific interval abandoned.
- Waited Less Then T - stores for each Customer Class an array of how many customers that arrived in a specific interval waited less then their defined threshold.
- Satisfied - stores for each Customer Class an array of how many customers that arrived in a specific interval and were admitted to service waited less then their defined threshold.

- Sojourns - stores for each customer class an array of how many customers that arrived in a specific interval spent time in the system less than their defined sojourn threshold.
- Waiting Times - stores for each Customer Class the summation of total waiting times of all customers that arrived in a specific interval.
- Wait Histograms - stores for each customer class an histogram of the total waiting time of all customers that arrived during the entire simulation horizon.
- Queue Length - stores for each Customer Class an array of the number in queue at the middle of specific interval.
- Number in System - stores for each pool an array of how many customers are in service in the middle of a specific interval.
- Utilization - stores for each customer class and for each pool the fraction of time spent on serving customers of the specific class out of the entire available time of all server in that pool.

Distribution This class is used to generate random variables according to given parameters and retrieve their moments. There are several types of random distribution implemented:

- $Exponential\,(\mu)$
- $HyperExponential\,(p_1, \mu_1, p_2, \mu_2)$
- $LogNormal\,(\mu, \sigma^2)$
- $Uniform\,(a, b)$
- $Normal\,(\mu, \sigma^2)$
- $Deterministic\,(value)$

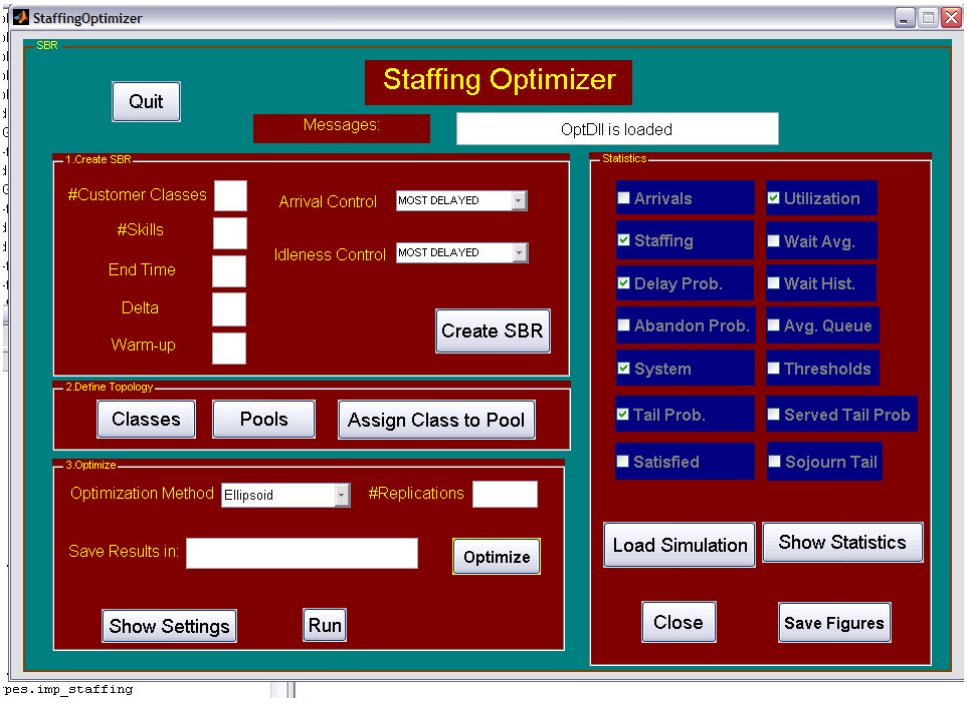
Image The Image class implements a "snapshot" of specific realization of the system at a given time. Images are used to save and restore the status of the system at a given time. This functionality is extremely useful when optimizing subsequent intervals in a time-varying settings. In this case the images are recorded at the end of each intervals, and by that the need to run simulation from the beginning of the period is prevented.

5.2 MATLAB GUI

System requirements: MATLAB v.6 or higher.
Installation requires only to unzip StaffingOptimizer.zip into a desirable directory. To run the GUI one must open a MATLAB session and run the command StsffingOptimizer from the relative path in which the StaffingOptimizer directory was placed. The tool GUI will open up. Note that the messages box should contain the text "SBRDll is loaded" - this indicates that the C++ simulation dll was loaded successfully.

5.2.1 Creating a Service Model

Figure 42: Main Screen of Staffing Optimizer GUI



In the main screen (see figure 42) one should define the following input:

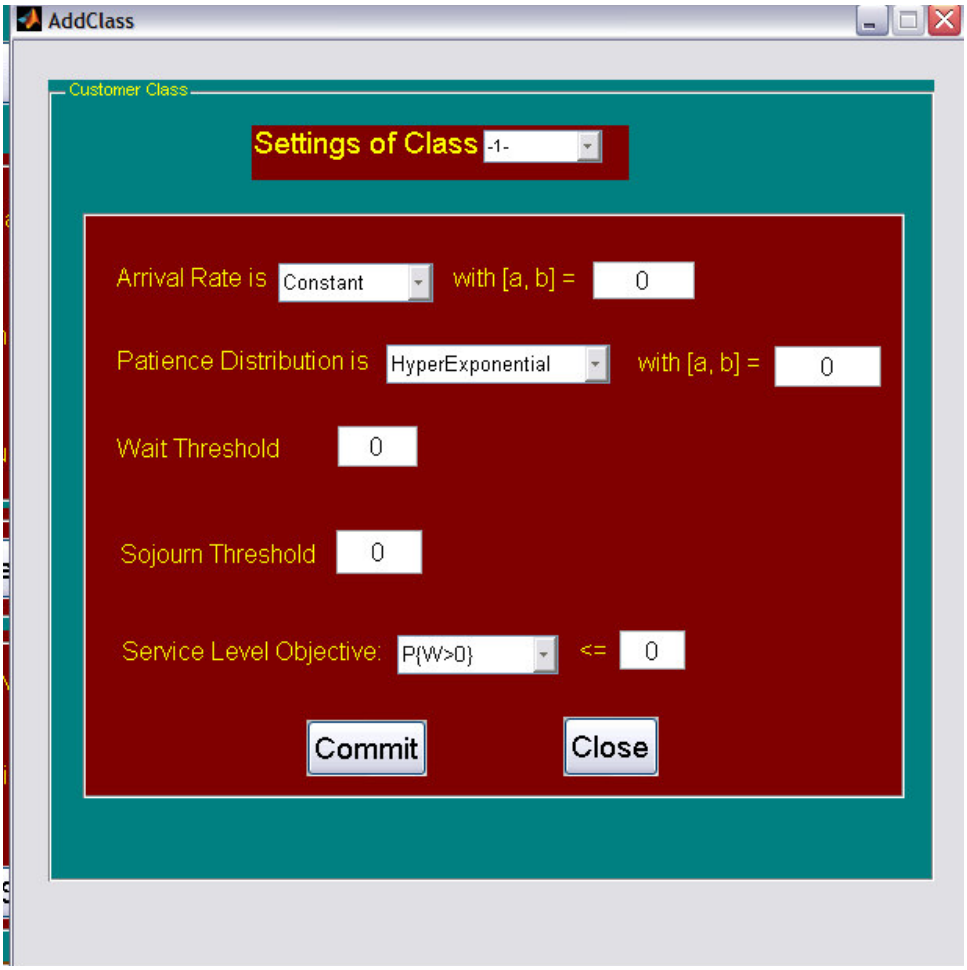
- Customer Classes - The number of customer classes
- Skills - The number of server pools
- End Time - the total simulation time

- Delta - the grid according to which statistics are gathered and displayed
- Warmup - The time from which statistics will be gathered
- Arrival Control - Choose the desired arrival control from a dropdown list.
- Idleness Control - Choose the desired idleness control from a dropdown list.

Then, the user should press the Create SBR button. The messages box should display the text: SBR Created. Pressing this button at any stage resets the system settings.

Definition of the customer classes is done by pressing the Classes button. When hitting

Figure 43: Classes Definition Screen

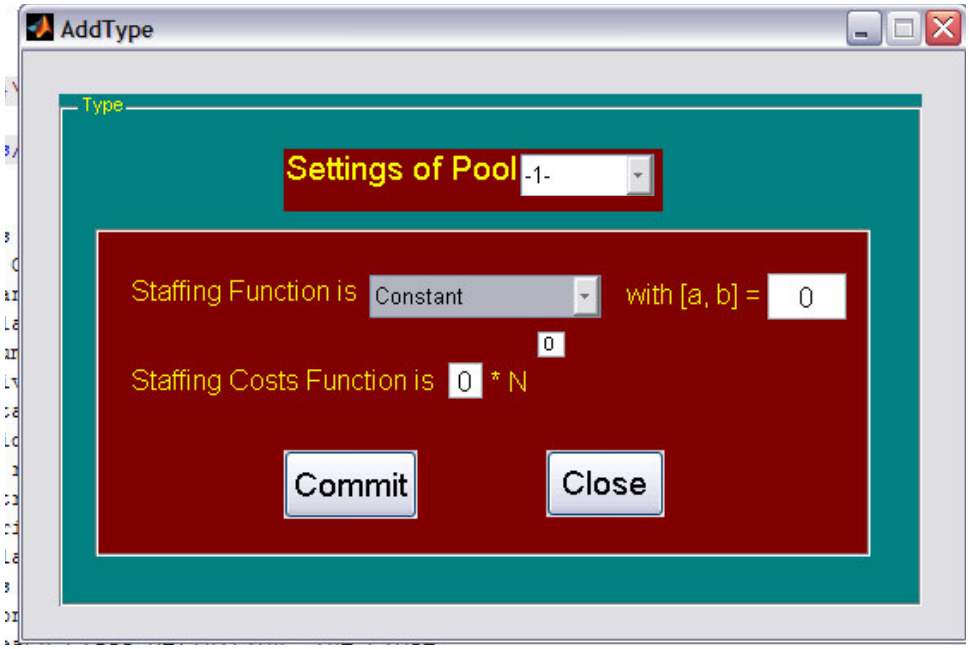


this button the Classes screen opens up (see Figure 43). The user can switch between the classes settings by choosing the desirable class from the top dropdown. The drop down

list is created with the number of classes that was fed in previous stage. In this form the arrival rate should be chosen. The option for this input is either constant - in this case one should only define a scalar rate, or sinusoidal - in this case one should input two scalars separated by space representing the formula $a + b \cdot \sin(t)$. Patience distribution with the parameters should be chosen. The available distribution types are detailed in simulation description (Statistics class). Wait and Sojourn thresholds are defined for statistics purposes. The last line is for optimization purposes and explained on 5.2.3. The Commit button should be pressed for each class definition. The Close button closes the screen.

Definition of servers pools is done by pressing the Pools button. Pressing this button opens up the Pools screen allowing to define the staffing levels and cost of each pool. The

Figure 44: Pools Definition Screen



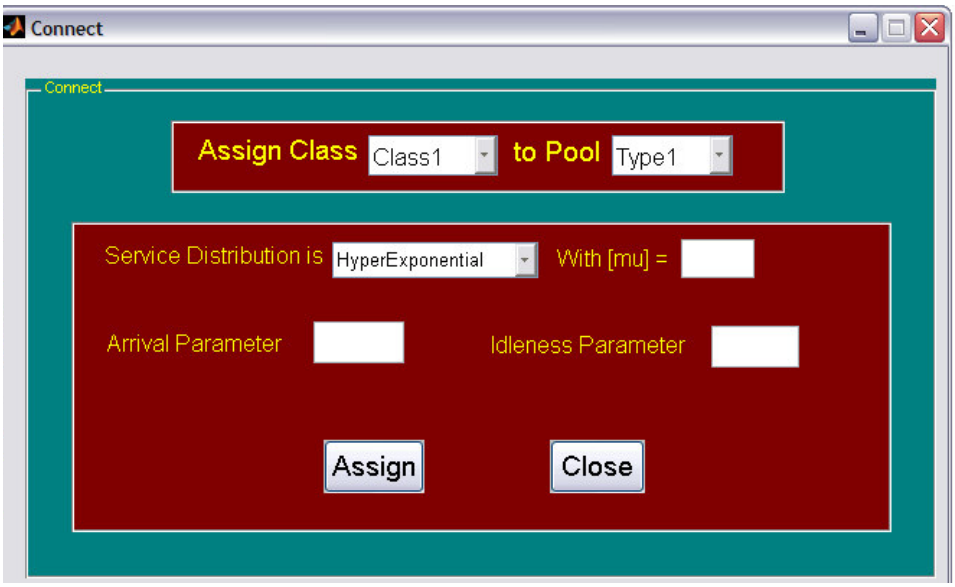
drop down list is created with the number of pools inserted in previous stage. There are three forms of staffing function definitions

- Constant - see arrival rate definition.
- Sinusoidal - see arrival rate definition.

- Functional - MATLAB style function inserted in the format of MATLAB function definition.

Recall that staffing levels can only vary from interval to interval and remain constant within each interval. The staffing levels in each interval will be evaluated as the function value at the beginning of the interval.
The cost definition will be explained on 5.2.3.
Pressing the Assign Class to Pool button opens up the connection screen that allows to define a connection between any class and any pool. The service distribution id defined

Figure 45: Connection Definition Screen

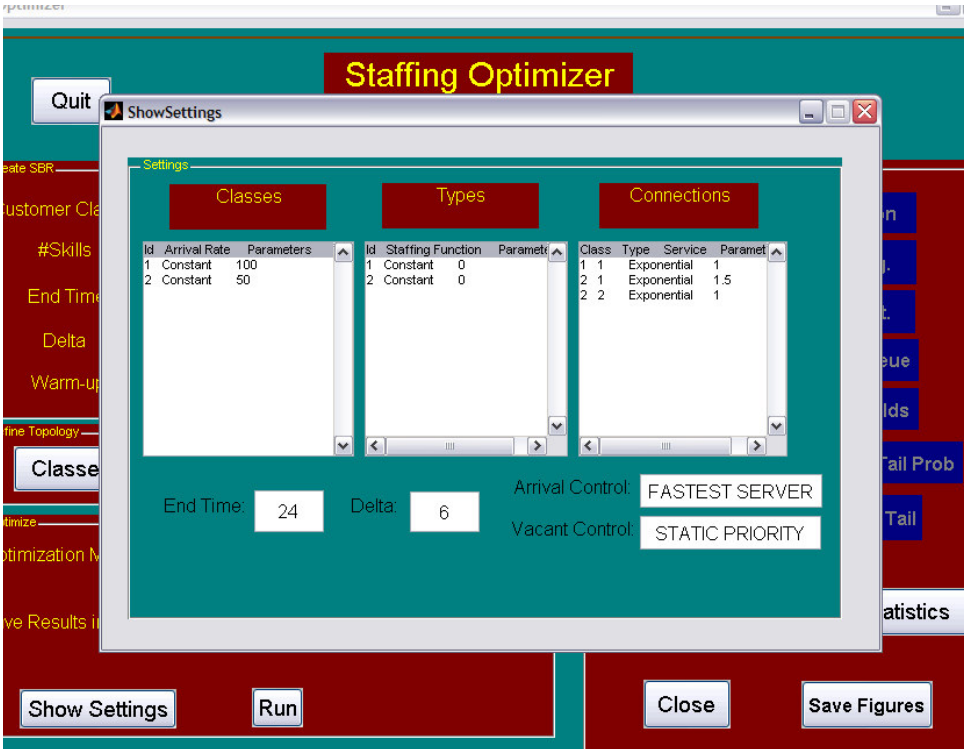


similar to patience distribution definition. In addition to control parameters can be defined for each class and pool, one for arrival control and one for idleness control.
Once completing these stages, the simulation model is properly defined. By pressing the Show Settings button, a window opens up displaying the system parameters including all classes pools and their connections.

5.2.2 Running Simulation and Viewing Results

After defining the simulation model (see 5.2.1), the following is needed in order to run the simulation:

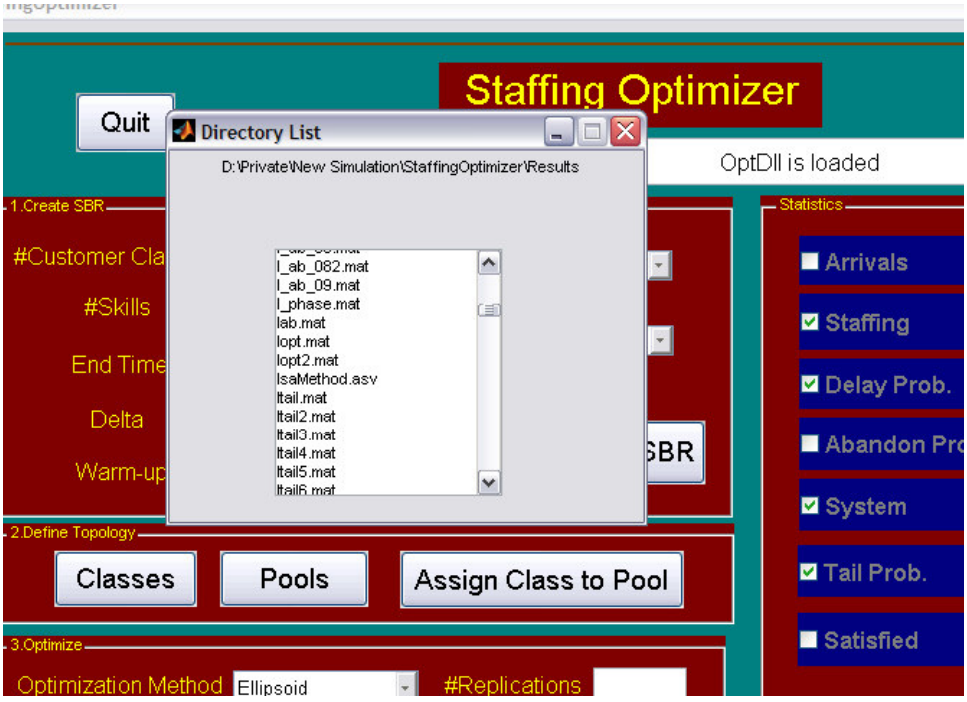
Figure 46: Summary Settings Screen



- Enter the number of replications the Replications text box
- Enter the name of the file to which results as well as model settings will be saved.
- Press the Run button.

During the simulation run, the message box displays the estimated time to complete the run. When done, simulation model and results are saved into file, and the message box should say "Results are saved in filename.mat". Any saved simulation can be loaded and viewed at a later time by pressing Load Simulation. A directory list is opened up (Figure 47). The user should double click the desired file from the results directory. To view

Figure 47: Loading Simulation Screen



the results the user should check all the statistics that he wishes to view. The available pre-interval statistics are:

- Arrivals - average of Arrivals statistics (see 5.2.2) over all replications
- Staffing - the staffing levels for each pool
- Delay Probability - sum of Delays statistics divided by Arrivals statistics.

- Abandonment Probability - the sum of Abandoned statistics divided by the sum of Arrivals statistics over all replications.
- System - the average number of each class in service of each pool and in queue.
- Tail Probability - the sum of Waited Less Than T statistics divided by the sum of Arrivals statistics over all replications.
- Satisfied - the sum of Satisfied statistics divided by the sum of Arrivals statistics over all replications.
- Utilization - the utilization profile of each pool (portion of time spend on each class and being idle)
- Wait Average - the sum of Waits statistics divided by the sum of Arrivals statistics over all replications.
- Wait Histograms - the waiting time histogram.
- Average Queue - the average queue of each class.
- Served Tail Prob - the Satisfied Statistics divided by Arrivals minus Abandoned statistics.
- Sojourn - the Sojourn statistics divided by Arrivals statistics.

Figure 5.2.2 displays sample statistics plots of arbitrary N model. Finally, pressing the Close button will all opened figures and pressing the Save Figures button will save all figures in a separate pdf file with the name of the statistics.

5.2.3 Running Optimization Algorithms

There are two optimization models available for use:

1. Cost Optimization - In this model, the service level is associated with some penalty cost.
2. Constraint Satisfaction - In this model, the service level appears as strong constraint.

Figure 48: Sample Plots of Simulation Results

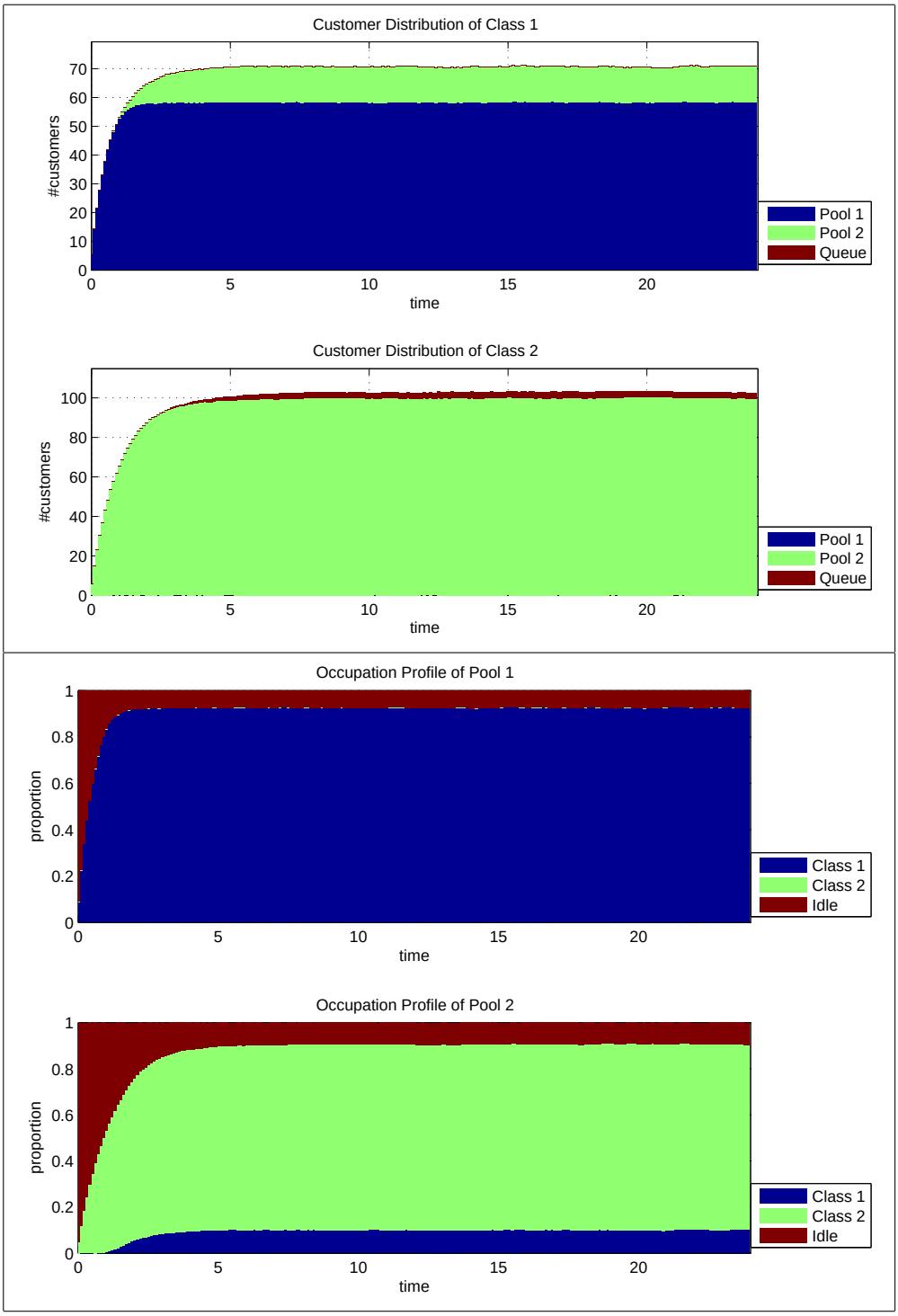
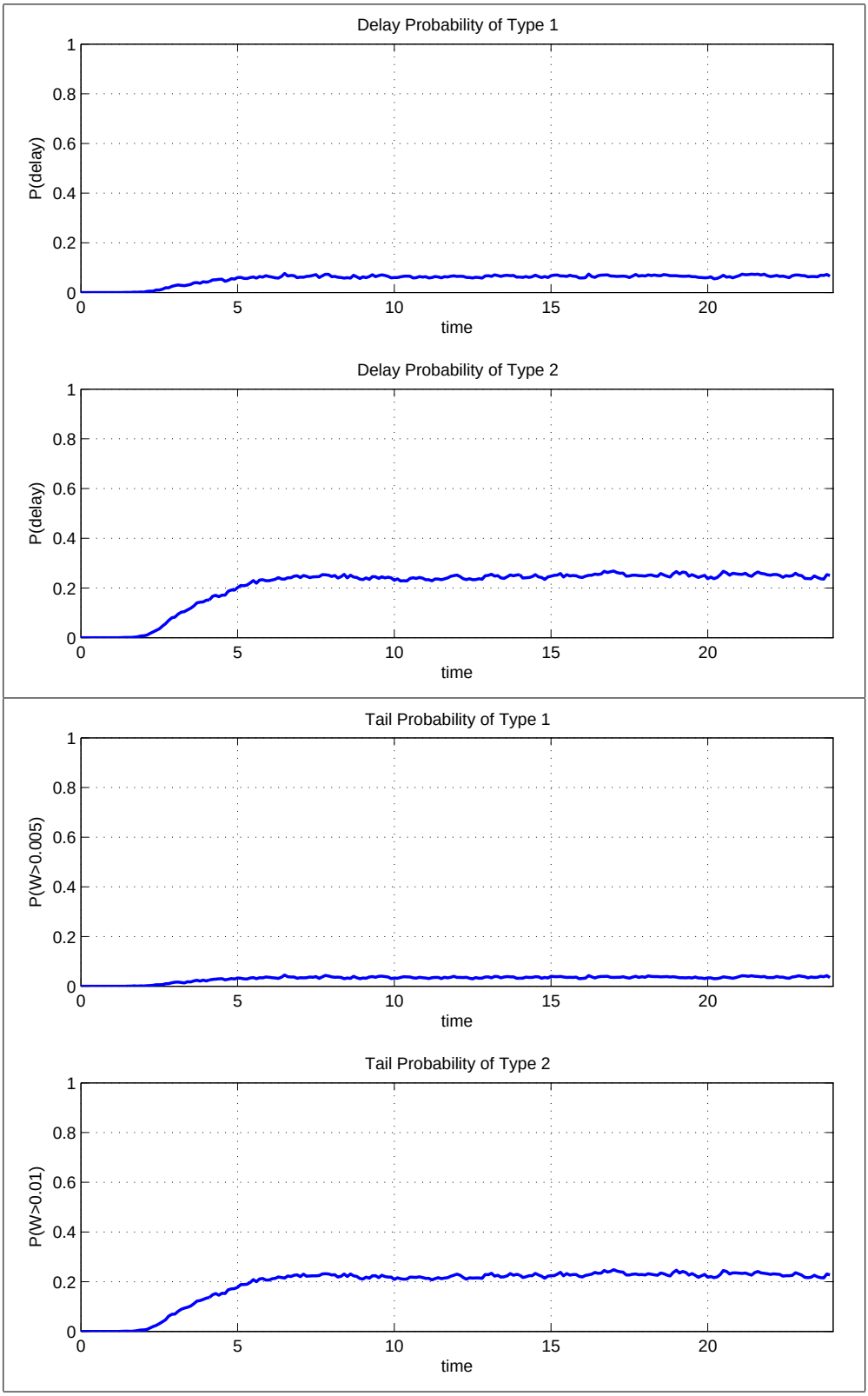


Figure 49: Sample Plots of Simulation Results



The penalty cost or constraint threshold are inserted in the classes screen. The desired service level objective should be picked up from a dropdown containing the following options:

- $\mathbb{P}\{W > 0\}$ - the delay probability.
- $\mathbb{P}\{W > t\}$ - the probability to wait more than threshold t. t is taken from the class tail threshold.
- $\mathbb{P}\{ab\}$ - the probability to abandon.
- $\mathbb{E}[W]$ - the expected wait.

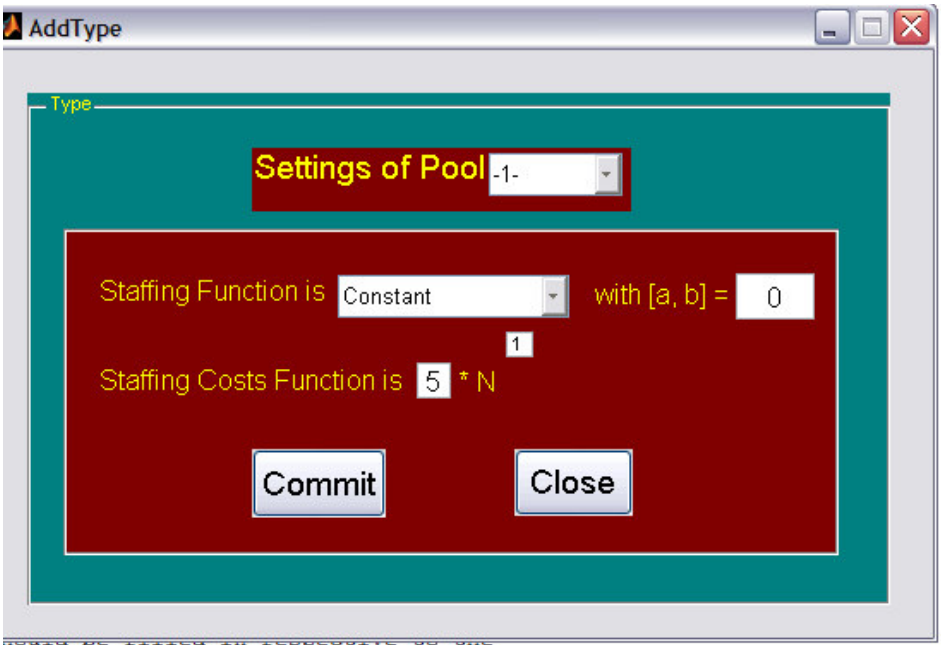
The cost or threshold should be filled in respective to the optimization model.

In addition, the cost of each pool should be defined in the Pools screen. In the this screen the user is able to edit the coefficients of polynomial cost function. Note that in the current version, the Cost Optimization mode allows only linear staffing function (i.e. the power component equals 1)

For instance, in Figure 50 there is a constraint defined for Class 1 stating that not more than 30% should wait more than 0.1 time units; The cost per time unit of Pool 1 servers is $5N^1$.

Finally, in order to run the optimization algorithm, one should choose the desired optimization method from the dropdown list and press the Optimize button. If the settings are time-varying, i.e. the chosen Delta is smaller than End Time implying multiple intervals, the algorithm will solve sequentially the problem for each interval. On the messages text box the currently solved interval will be displayed. Once solution is reached, a simulation will run with the obtained staffing. The results will be saved to file and will be displayed automatically.

Figure 50: Defining Optimization Parameters



6 Future Work

In this work we developed algorithms for identifying optimal staffing of systems with skills-based-routing given the routing scheme. These algorithms can be used for very general settings, including time-varying models and general distributions, to either optimize service levels and labor costs or optimize labor costs while satisfying some desirable Service Level constraints.

The algorithms were proven to work very well, and in most cases attained the optimal solution even when the service levels were not convex in the staffing levels. However, in few other cases, the non-convexity nature of the service levels caused the algorithms to converge to sub-optimal solution. Although this can be controlled by applying simple procedures, it can be interesting to try to find alternative algorithms that do not rely on the convexity, and might even work faster.

While our solution can be very practical for many applications, it lacks the ability to incorporate labor scheduling constraints and thus can not be used to provide an optimal scheduling solution.

One important direction is to develop algorithms or combine the ones that are introduced in this work, with a scheduling mechanism which is traditionally carried out by some mathematical programming solver.

Another potential direction is to provide optimal solution for more complex systems with complex processes, such as Service Networks, Petri Nets etc. This will probably require more sophisticated algorithms.

References

[1] Aksin, Z., Armony, M., Mehrotra, V., **The modern Call-Center: A Multi-Disciplinary Perspective on Operations Management Research**. *Production and Opetations Management*, November 2007 1.1

[2] Armony, M. and Mandelbaum, A. **Design, Staffing and Control of Large Service Systems: The Case of a Single Customer Class and Multiple Server Types**. Draft, March 2004. 1.1, 1.1

[3] Atlason, J., Epelman, E., Henderson, S.,**Call-Center Staffing with Simulation and Cutting Plane Methods** 1.1, 3, 4.4

[4] Bassamboo, A., Randhawa, R. S., **A Little Flexibility is All You Need – Optimality of Tailored Chaining** 1.1

[5] Eick, S., Massey, W., Whitt, W. **The Physics of The $M_t/G/\infty$ Queue.** *Operations Research*, **41**(4), 731-742, 1993.

[6] Feldman, Z., Mandelbaum, A., Massey, W.A., Whitt, W. **Staffing of Time-Varying Queues to Achieve Time-Stable Performance**, *Management Science*, 2006.

[7] Gans, N., Koole, G., Mandelbaum, A. **Telephone Call Centers: Tutorial, Review and Research Prospects**. *Invited review paper by Manufacturing and Service Operations Management* (MSOM), 5 (2), pp. 79–141, 2003. 1, 1.1

[8] Garnett, O., Mandelbaum, A. **An Introduction to Skills Based Routing and its Operational Complexities**. May 2000.

[9] Garnett, O., Mandelbaum, A. and Reiman, M. **Designing a Call Center with Impatient Customers**. *Manufacturing and Service Operations Management*, **4**(3), 208–227, 2002.

[10] Glynn, P.W **Limit theorems for the method of replications** *Comm. ACM 33, 75-84* 3

[11] Gurvich, I., Armony, M. and Mandelbaum, A. **Service Level Differentiation in Call Centers with Fully Flexible Servers**. Draft , April 2006. 1.1, 4.1, 4.1

[12] Gurvich, I., Whitt, W., **Service Level Differentiaition in Many-Server Service Systems: A Solution Based on Fixed-Queue-Ratio Routing** 1.1, 4.2.2

[13] Gurvich I., Whitt W. (2006) **Asymptotic Optimality of Queue-Ratio Routing for Many-Server Service Systems**. Submitted, September 2008 1.1

[14] Halfin, S., Whitt, W. **Heavy-Traffic Limits for Queues with Many Exponential Servers**. *Oper. Res., 29(1981), 567-587.*

[15] Juditsky, A., Lan, G., Nemirovski, A., Shapiro, A. **Stochastic Approximation Approach to Stochastic Programming** 1, 2, 1.2, 2.2, 2.2, 2.2, 2.2, 3, 3.1, 3.2

[16] Nemirovski, A., Yudin, D., **Problem complexity and method efficiency in optimization**, *Wiley- Interscience Series in Discrete Mathematics, John Wiley, XV, 1983*

[17] Polyak, B.T. and Juditsky, A.B., **Acceleration of stochastic approximation by averaging**, *SIAM J. Control and Optimization, 30 (1992), 838-855* 2.2

[18] Robbins, H. and Monro, S., **A Stochastic Approximation Method**, *Annals of Math, Stat., 22 (1951), 400-407* 2.2

[19] Wallace, R.B., Whitt, W. **A Staffing Algorithm For Call Centers With Skill-Based Routing**, August 2004. 1.1, 1.1

[20] Y.C. Ho, X.R. Cao **Optimization and perturbation analysis of queueing networks** *J. Optim. Th. Appl. 40, 559-582*

3