UNIVERSITY OF CALIFORNIA,
IRVINE

Routing Dynamics: Optimization, Measurement, and Applications

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Management

by

Vahid Nourbakhsh

Dissertation Committee:
Associate Professor John G. Turner, Chair
Professor L. Robin Keller
Professor Rick (Kut) So

2018

DEDICATION

In dedication to my supportive wife Hoda, and my family who value science and research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my advisor, Professor John Turner, who guided me throughout this journey, my committee, Professors Robin Keller and Rick So, the Operations and Decision Technologies faculty, Professors Shuya Yin, Luyi Gui, and Carlton Scott, my fellows in the Ph.D. program, and all persons who shared their insights and their comments, and helped me in my research.

CURRICULUM VITAE

Vahid Nourbakhsh

EDUCATION

Doctor of Philosophy in Management **2018**
University of California, Irvine *Irvine, California*

Master of Science in Systems Engineering **2012**
Amirkabir University of Technology *Tehran, Iran*

Bachelor of Science in Industrial Engineering **2009**
Amirkabir University of Technology *Tehran, Iran*

RESEARCH EXPERIENCE

Graduate Research Assistant **2013–2018**
University of California, Irvine *Irvine, California*

TEACHING EXPERIENCE

Teaching Assistant **2013–2018**
University of California, Irvine *Irvine, California*

REFEREED JOURNAL PUBLICATIONS

Govindan, K., Jafarian, A., and Nourbakhsh, V. (2015). Bi-objective integrating sustainable order allocation and sustainable supply chain network strategic design with stochastic demand using a novel robust hybrid multi-objective metaheuristic. Computers & Operations Research, 62, 112-130.

Devika, K., Jafarian, A., and Nourbakhsh, V. (2014). Designing a sustainable closed-loop supply chain network based on triple bottom line approach: A comparison of metaheuristics hybridization techniques. European Journal of Operational Research, 235(3), 594-615.

Nourbakhsh, V., Ahmadi, A., and Mahootchi, M. (2013). Considering supply risk for supplier selection using an integrated framework of data envelopment analysis and neural networks. International Journal of Industrial Engineering Computations, 4(2), 273-284.

ABSTRACT OF THE DISSERTATION

Routing Dynamics: Optimization, Measurement, and Applications

By

Vahid Nourbakhsh

Doctor of Philosophy in Management

University of California, Irvine, 2018

Associate Professor John G. Turner, Chair

We study the problem of routing jobs in multi-class multi-server queueing systems, where the service rate depends both on the job type and the server type. This queueing setting arises in different applications. For example, in transportation systems, a vehicle's travel time depends on the job's location (e.g., a fire incident's location) and the vehicle's location (e.g., a fire station). In call centers, an agent's service time depends on her skills and the call type that is routed to her.

Chapter I focuses on maximizing the service level of jobs defined as the probability that a job is served within an acceptable waiting time. Service level is an important performance measure for queueing systems. For example, emergency vehicle networks should make sure that the selected server (e.g., a fire engine) arrives at the requester's location within the specified acceptable waiting time. We employ a mathematical programming approach which is desirable since it can easily be embedded within larger planning problems such as determining the optimal location of vehicle hubs and finding the minimum number of vehicles for the desired coverage level. We show that the expected waiting time is convex with respect to its variables, namely, arrival rate and workload. For this reason, the expected waiting time objective is more common in the literature, even though in many applications it is the service level function which is more practical. The service level function, however, is non-convex

and requires more advanced methods to solve for optimal or near-optimal solutions. We develop one such advanced method, the Fixed-ratio Shifting Envelopes (FSE) method. Our extensive numerical experiments indicate that FSE outperforms other benchmark algorithms as well as the global solver BARON. In Chapter I, we end our treatment of the subject with the static policy developed for this novel math programming formulation.

Chapter II seeks online (or, real-time) routing policies that answer two questions: (i) Upon a job arrival, to which server group should we assign that job? and (ii) When a server becomes free, which job should the server begin to serve? We point out that the optimal dynamic policy for minimizing the expected waiting time or maximizing the service level is not characterized in the literature. In Chapter II we focus on designing a dynamic policy for the widely-used measure of minimizing the expected waiting time. We develop a math program to model a static variant of our routing problem. Then, we use the solution from our math program to build novel dynamic policies. Our experiments show that one of our proposed overflow dynamic routing policies, which we call FSFOptXOverflowBlock, outperforms *Fastest-Server-First*, a well-known routing policy for such problems in practice and in the literature. To showcase our methodology, we apply our proposed policy to the problem of assigning fire incidents in Irvine, CA, to fire stations. Methodologically, one could extend this so-called dynamization technique to also construct dynamic policies for the service level maximization problem of Chapter I. We leave the details for future research.

Chapter 1

Maximizing the Service Level of Jobs in Multi-Class Multi-Server Queueing Systems: An Iterative Heuristic Approach

1.1 Introduction

We consider a routing problem for a multi-class multi-server system with Poisson arrivals and service rates that depend on both the job and the server. Each group of servers has a single queue, and servers within a group are homogeneous and have exponentially distributed service times that depend on the job type. We call these service times job-and-server dependent rates.

This routing problem arises in different applications including allocating vehicles to requesters in *Transportation-On-Demand* (TOD) systems, routing calls to agent groups at

call centers, and allocating user tasks to distributed processors (also known as *load balancing*).

In TOD systems such as emergency systems (i.e., ambulances, fire trucks, police patrols, etc.), courier services and taxi networks, the system randomly receives service requests. For each request, a server (i.e., an ambulance, fire truck, police patrol, courier or taxi) is dispatched to the requester's location, serves the requester, finishes the service, returns to its base, becomes available and waits for the next request. In this setting both requesters and vehicles are geographically distributed. This in turn gives rise to job-and-server dependent service rates where the service rate depends on the distance between the vehicle base and the requester (c.f., Cho *et al.* 2014). For an excellent survey on TOD, please refer to Cordeau *et al.* (2007).

In a call center, calls of different types arrive randomly and a specialized switch called an Automatic Call Distributor (ACD) routes them to agents. Agents within different groups have different skill sets and consequently different service rates for serving a job, which makes the router's decision challenging. There is a plethora of research on skill-based routing for call centers (for an excellent survey see Gans *et al.* 2003).

In computer systems, a dispatcher distributes jobs generated by users over a set of processors. *Load balancing* has been well studied in the literature (c.f., Combé & Boxma 1994). Job-and-server dependent service times can occur for two different reasons, i) the processors are geographically distributed and the service time is the sum of transferring the job to a processor and the processing time spent on the processor, ii) when the processors are of different speeds and jobs are of different types.

One of the most important performance measures for evaluating the aforementioned queueing systems is *Service Level* (SL), i.e., the probability that a server begins the job's service within an *Acceptable Waiting Time* (AWT). In TOD systems, there is an acceptable waiting time

which is the time that a vehicle arrives at the requester's location or the time it takes to arrive at the requester's location, pick her up and drop her off at the destination. In North America, reaching 90% of urgent urban emergency medical services (EMS) calls in 9 minutes is a common target (Fitch, 2005). The National Health Service in the United Kingdom mandates ambulances to reach 75% of life-threatening calls in 8 minutes and 95% of all calls in 19 minutes (Department of Health, 2015). Similarly, the National Fire Protection Association (2004) in the United States recommends that with a probability of at least 90% a fire engine must arrive at a building on fire within 4 minutes. At call centers, policy-makers decide on a target for the acceptable waiting time to make sure that customers do not wait too long for agents to answer their calls. Typically the target is to answer 80% of calls in 20 seconds (Gans *et al.*, 2003; Cheong *et al.*, 2008; Pot *et al.*, 2008). SL as defined below captures these objectives, which are important in practice,

$$SL := Pr[\text{Waiting Time } \leq \text{Accepable Waiting Time}]$$

Other performance measures include *expected waiting time*, i.e., the expected time in the queue, *expected sojourn time*, i.e., the expected time in the system including queue waiting time and service time, and *expected throughput*, i.e., the expected number of jobs that are not blocked and are served. *Service level*, which is commonly what decision makers actually want to measure, is seldom optimized due to its non-convexity; whereas *expected waiting time*, *expected system time* and *expected throughput* are convex and so more widely studied and used in practice.

Our main contributions are as follows. We formulate a general math program to optimize key routing variables that apply when maximizing SL. Then, we investigate the convexity of this math program and show that it is neither convex nor pseudo-convex. Borrowing key concepts from the *Shifting Quadratic Envelopes* algorithm (SQE) of Cho *et al.* (2014), we

develop a tailored algorithm called *Fixed-ratio Shifting Envelopes* (FSE) to solve our non-convex SL-maximizing math program to near-optimality. FSE is a heuristic which exploits key structures inherent in our SL maximization problem. Moreover, we prove that in theory the underlying math program can approximate the original non-convex one to arbitrary precision. In our numerical experiments we show that FSE significantly outperforms SQE. In particular, our boundary generation heuristic, which is significantly more advanced than that of SQE, leads to FSE significantly outperforming SQE on SL-maximization problems. We conduct a comprehensive set of numerical experiments and show that FSE outperforms not only SQE, but also the commercial solvers BARON and KNITRO, as well as other FSE-related algorithms that we introduce to illustrate the importance of FSE's key features.

The rest of this paper is organized as follows. Section 1.2 reviews related literature. In Section 1.3, we describe the model and examine problem convexity. Section 1.4 describes the solution algorithms FSE and SQE. In Section 1.5 we conduct computational experiments to assess our solution algorithms. Section 1.6 introduces extensions to our model and FSE algorithm that apply to SL constraints rather than a SL objective. We conclude in Section 1.7.

1.2 Literature Review

There are two categories of routing policies in the literature: static and dynamic. Static policies use characteristics of the system such as the arrival/service rate and distribution. Dynamic policies consider the system status when each job arrives (e.g., the number of jobs in the system/queues and whether particular servers are busy). By design, our model assumes exponential arrival and service rates with known parameters, and produces a static policy.

It is clear that, in general, dynamic policies perform better than static policies. However,

static policies are also of considerable interest. First, unlike dynamic policies, static policies do not depend on real-time information about the system's state, which may be costly or impractical to track in some applications.

Second, static policies are useful tools for designing a queueing system. Static policies are generally faster to compute and optimize whereas dynamic policies are often hard to analyze, and for this reason their performance under a given arrival rate and workload is often estimated via time-consuming methods such as simulation. Design problems that static policies can be used for include, among others, location of vehicle bases in TOD systems (c.f. the location-allocation problem in Louwers *et al.* 1999) and the queue design problem for determining the number of servers for each group, also known as the staffing problem for call center planning (c.f., Atlason *et al.* 2008 and Cezik & L'Ecuyer 2008).

Third, while most of the research on dynamic policies concerns a single job type, the optimal dynamic policy for routing jobs of heterogeneous types to heterogeneous server groups is not known (Mehrotra *et al.*, 2012). Armony (2005) has shown that the *Fastest-Server-First* (FSF) dynamic policy asymptotically minimizes the steady-state expected waiting time in the Halfin-Whitt many-server heavy-traffic regime also known as the Quality and Efficiency Driven (QED) regime. Upon a job arrival or service completion, FSF routes the job at the head of the queue to the fastest server available. However, the result is limited to a single job type and the optimality of FSF has not been shown for a multi-class multi-server system with heterogeneous job types and heterogeneous server groups. Dai & Tezcan (2008) provide some insight into the challenge of proving such a result, which suggest that FSF is not optimal in this general setting.

There is a body of research that focuses on finding optimal routing policies and we review some of the most relevant ones. Ephremides *et al.* (1980) consider a simple model in which, as jobs arrive, they are routed to one of two similar exponential servers. They show that if the queue lengths at both servers are observed then the optimal decision is to route jobs to

the shorter queue, whereas if the queue lengths are not observed then it is best to alternate between queues, provided the initial distribution of the two queue sizes is the same. The optimality of these routing strategies is independent of the distribution of the job arrivals.

Hordijk & Koole (1990) generalize the model introduced by Ephremides *et al.* (1980) to the case with multiple servers. Arriving customers choose between multiple parallel servers, each with its own queue. For general arrival streams, they prove that the policy which assigns customers to the shortest queue is stochastically optimal for models with finite buffers and batch arrivals. Their result is limited to one job type and homogeneous server groups that each have exactly one server.

Liu & Towsley (1994) consider the problem of routing customers to identical servers, each with its own infinite-capacity queue. Under the assumptions that (i) the service times form a sequence of independent and identically distributed random variables with increasing failure rate distribution and (ii) state information is not available, they establish that the round-robin policy minimizes the customer response times and the numbers of customers in the queues.

Combé & Boxma (1994) study static routing policies for single-class multi-server systems, where jobs arrive according to a Poisson process and the service times are independent random variables with known first and second moments. They build two classes of static policies, namely, *probabilistic assignment* and *allocation according to a fixed pattern*, by solving a convex mathematical program, which optimizes performance measures such as expected waiting time and expected queue length. For servers with exponential service times, we extend their results to the multi-class multi-server case.

Koole (1999) studies the static assignment to parallel, exponential, heterogeneous servers. Blocked customers are lost, and the objective is to minimize the expected number of blocked customers. The problem is formulated as a stochastic control problem with partial observa-

tion, and an equivalent full observation problem is formulated.

Gans & Zhou (2003) study the routing of two job types (i.e., high priority and low priority) to a pool of homogeneous servers. A system controller seeks to maximize the rate at which low priority jobs are processed, subject to service quality constraints (i.e., service level or expected waiting time) on high priority calls. They characterize optimal routing policies for the special case where both call types have equal expected service times. They also show that in the general case of jobs with different expected service times, the policies are optimal within the class of priority policies.

Our problem which maximizes the non-concave SL objective for heterogeneous job types and heterogeneous servers is more complex than much of the literature which considers expected waiting time. The service level problem, although non-convex, is an important measure in practice that fits many applications better than the expected waiting time measure, which is convex (c.f., Pichitlamken *et al.*, 2003; Gans *et al.*, 2003, for the importance of the SL measure in call centers). Our problem, due to its representation as a math program, can be embedded into location-allocation problems to determine the location of hubs in TOD systems, or can be embedded into staffing and scheduling problems such as those in call centers. Finally we point out that the resultant static policy from solving our math program can be translated into a dynamic policy. In particular, in Nourbakhsh & Turner (2018) we show how to build several so-called "dynamized" policies from a static policy meant to minimize expected waiting time, and through simulation experiments show that some such dynamized policies beat the commonly-used dynamic policy Fastest-Server-First.

1.3 Model

Jobs of types $i \in I$ arrive according to independent Poisson processes with rates d_i. Each job must be routed to a server group $j \in FJ_i$, where FJ_i is the set of server groups that are eligible to serve jobs of type i. For completeness, we will denote J as the full set of server groups (henceforth known simply as groups) and FI_j as the set of job types that can be served by group j. In the call center literature, the terms job type and server group are referred to as call type and agent group, respectively. The number of servers in each group is a known parameter k_j. We will call the model single-server if $k_j = 1,\ \forall j \in J$, and multi-server otherwise. The model is schematically depicted in Figure 1.1 with notation summarized in Table 1.1.

For each queue j, we assume that (i) we deploy static routing to route jobs from job types to groups; (ii) service rates are independent, each exponentially distributed with mean service time τ_{ij}; (iii) the service discipline at each group is first-come, first-served; and (iv) the system backlogs jobs at the groups until there are available servers to serve the jobs, i.e., no abandonment or retrial.

The model preserves Poisson arrivals at each group, thus queues at groups are M/M/k and the delay probability at group j is computed using the Erlang-C function (Cooper, 1981) defined as (for simplicity we suppress group index j),

$$C(k,r) = \frac{r^k}{(k-1)!(k-r)} \times \Big[\sum_{n=0}^{k-1} \frac{r^n}{n!} + \frac{r^k}{(k-1)!(k-r)}\Big]^{-1}. \tag{1.1}$$

We expect, however, that our model would also be useful in the context where service times are not exponential since others (c.f., Kimura 2010) have shown the Erlang-C function tends to be a good approximation for the delay of a M/G/k queue, which has no known closed-form formula.

Table 1.1: Model notation

Indices and Sets	
$i \in I$	Index for job types
$j \in J$	Index for server groups
F	Set of feasible job type-server group assignments ($i \to j$)
FJ_i	The subset of groups that can serve jobs of type i
FI_j	The subset of job types that group j can serve
Parameters	
d_i	Expected arrival (demand) rate of job type i
τ_{ij}	Mean service time for a server at group j to serve a job of type i
k_j	Number of servers at group j
T	Acceptable waiting time in the queue
C_f	A number between zero and one indicating the minimum fraction of jobs that should be covered (routed)
Variables	
x_{ij}	Number of jobs of type i to be served by group j per unit time
λ_j	Total number of jobs to be served by group j per unit time
r_j	Workload assigned to group j
τ_j	Mean service time at group j

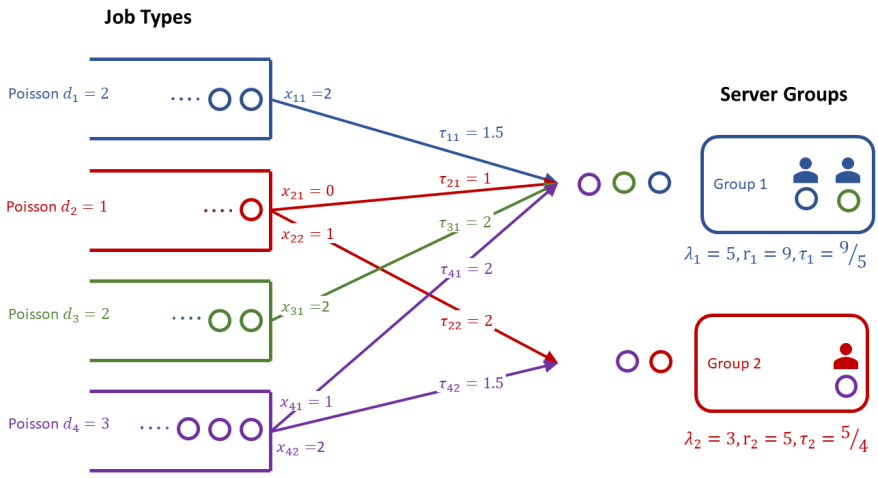


Figure 1.1: Schematic representation of the multi-class multi-server model.

The decision variable x_{ij} determines the number of jobs of type i to be served by group j per unit time. The arrival rate to group j is the sum of all jobs of all types that it serves,

$$\lambda_j = \sum_{i \in FI_j} x_{ij} \quad \forall j \in J. \tag{1.2}$$

The workload of group j is

$$r_j = \sum_{i \in FI_j} \tau_{ij} x_{ij} \quad \forall j \in J. \tag{1.3}$$

The mean service time at group j is defined as

$$\tau_j := \frac{r_j}{\lambda_j} \quad \forall j \in J. \tag{1.4}$$

1.3.1 Objective Function: Service Level

We seek to determine the routings, i.e., the number of jobs of each type i to be served by group j per unit time, and represented by the variables x_{ij} for each edge (i, j) in Figure 1.1, such that the service level that jobs experience is maximized. The *Service Level* (SL), also known as the *Telephone Service Factor* (TSF) for call centers, is the probability that a job is served within an acceptable waiting time T,

$$\begin{aligned} SL &:= P[W \leq T] \\ &= 1 - P[W > 0]P[W > T|W > 0], \end{aligned} \tag{1.5}$$

where W is the random waiting time in the queue. The model assumptions allows us to use the Erlang-C function defined in Equation 1.1 to compute the delay probability, $P[W > 0]$. Given the event that an arriving job must wait for a server, the conditional delay $P[W > T|W > 0]$ is exponentially distributed with mean $\big(k - r/\tau\big)^{-1} = \big(\lambda(k-r)/r\big)^{-1}$ (c.f. Abate *et al.* 1995 and Gans *et al.* 2003). Thus from equation (1.5), SL becomes,

$$SL(\lambda, r, k, T) = 1 - C(k, r)e^{-\left(\frac{\lambda(k-r)}{r}\right)T}. \tag{1.6}$$

For representation conciseness we write the service level as a function of variables λ and r, and omit parameters k and T, i.e., we denote $SL(\lambda, r, k, T)$ as $SL(\lambda, r)$.

Figure 1.2 shows a service level function with three different acceptable waiting time values. Notice the sensitivity of the SL function to workload r and acceptable waiting time T. It is this difference between the functional form of SL and Erlang-C (i.e., SL with acceptable waiting time of zero) that we exploit in our solution algorithm FSE. In Section 1.4 we explain our FSE in detail and delineate how FSE takes advantage of the functional form of SL to numerically outperform SQE, the original method introduced by Cho *et al.* (2014).

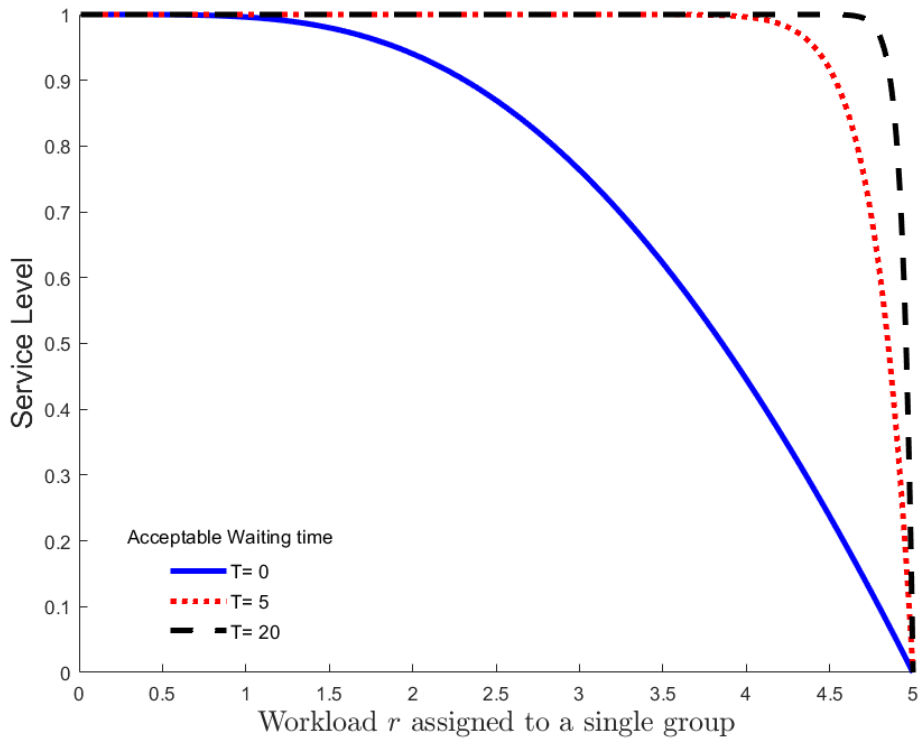


Figure 1.2: Service Level as a function of workload with $\lambda = 4$ arrivals and $k = 5$ servers. Note that the Service Level function with acceptable waiting time $T = 0$ reduces to the Erlang-C function.

The function in (1.6) computes the service level for a single group (or equivalently a single queue before each group). For the queue in front of server group j, the service level is $SL(\lambda_j, r_j)$, where λ_j and r_j are the arrival rate to group j and the workload of group j, respectively.

In the multi-class multi-server system which is the focus of this paper, we are interested in aggregate measures that can help the decision maker evaluate the performance of the whole system, not just a single group. For this purpose, we introduce two aggregate measures, namely, the *total service level* and the *average service level* denoted by SL_{tot} and SL_{avg}, respectively. We define SL_{tot} across all groups as,

$$SL_{tot} := \sum_{j \in J} \lambda_j SL(\lambda_j, r_j), \tag{1.7}$$

where SL_{tot} measures the total number of jobs served within the desired acceptable waiting time. We note that SL_{tot} often has a useful application-specific interpretation.

An alternative to the total service level function, SL_{tot}, is the average service level function, SL_{avg}, defined as the service level that a random arriving job of any type experiences,

$$SL_{avg} := \frac{\sum_{j \in J} \lambda_j SL(\lambda_j, r_j)}{\sum_{j \in J} \lambda_j}. \tag{1.8}$$

In the following subsection we formally define the math program (P) with the service level objective, i.e., SL_{tot} or SL_{avg}.

1.3.2 Math Program

The math program (P) below determines routings x_{ij}'s such that the service level is maximized.

$$
\begin{aligned}
(P) \quad \max \quad & SL_{tot} \text{ or } SL_{avg} \\
\text{s.t.} \quad & \text{Constraints (1.2) and (1.3)}, \\
& r_j \le k_j \quad \forall j \in J, && (1.9) \\
& \sum_{j \in FJ_i} x_{ij} \le d_i \quad \forall i \in I, && (1.10) \\
& \text{Optional convex constraints}, && (1.11) \\
& x_{ij} \ge 0 \quad \forall (i,j) \in F. && (1.12)
\end{aligned}
$$

Constraint (1.9) assures the workload at each group does not exceed the number of servers at that group, and is required for the system to be in steady state. Constraint (1.10) makes sure that for each job type i, the covered jobs are less than or equal to the arrival rate (also called demand) at node i. Constraint (1.10) also indicates that not all jobs are necessarily served and some are possibly blocked upon arrival, i.e., we determine allocation and coverage simultaneously. To preserve Poisson arrivals at the group queues, this blocking is done randomly proportional to $\{x_{ij}\}$. Constraint (1.11) indicates that one has the option to impose convex constraints on x_{ij}'s, r_j's and λ_j's. For example, we can make sure that the workload is roughly balanced between groups by imposing lower and upper bounds on r_j's. Or, we can easily embed (P) into a much larger math program to link additional decisions. Constraint (1.12) makes sure that x_{ij}'s are nonnegative.

When optimizing SL_{tot}, there is an endogenous trade-off between an increase in the number of served jobs (i.e., an increase in $\sum_{j \in J} \lambda_j$) and an increase in the queues' congestion and the resulting service delays that adversely affect the SL_{tot} objective. Thus, coverage and

routing can be simultaneously optimized. However, for SL_{avg} this trade-off does not exist and without enforcing a minimum coverage constraint, the optimal solution will be degenerate, have no congestion, and not serve any job. Thus, for SL_{avg} we exogenously enforce a minimum coverage level using the constraint,

$$\sum_{j \in J} \lambda_j \geq C_f \sum_{i \in I} d_i, \tag{1.13}$$

where $0 \leq C_f \leq 1$ is the minimum (or, desired) coverage factor.

Proposition 1.3.1 proves that constraint (1.13) is binding at optimality. Indeed, one cannot improve SL_{avg} or SL_{tot} by increasing the coverage, $\sum_{j \in J} \lambda_j$, beyond the minimum required coverage, $C_f \sum_{i \in I} d_i$.

Proposition 1.3.1. *Assuming that problem (P) with the coverage constraint (1.13) is feasible, the coverage constraint (1.13) is binding for the SL_{avg} objective.*

Proof. See Appendix A. □

Proposition 1.3.1 indicates that at optimality SL_{avg} becomes

$$SL_{avg} = \frac{\sum\limits_{j \in J} \lambda_j SL(\lambda_j, r_j)}{\sum_{j \in J} \lambda_j} = \frac{\sum\limits_{j \in J} \lambda_j SL(\lambda_j, r_j)}{C_f \sum_{i \in I} d_i} = \frac{SL_{tot}}{C_f \sum_{i \in I} d_i}. \tag{1.14}$$

Since $C_f \sum_{i \in I} d_i$ is a constant, optimizing SL_{avg} in Problem (P) with Constraint (1.13) is equivalent to optimizing SL_{tot} in problem (P) with Constraint (1.13). We will therefore focus on the SL_{tot} objective for the remainder of the paper.

1.3.3 Lack of Concavity of SL_{tot}

In this subsection we investigate the joint concavity of the objective function SL_{tot} in λ_j and r_j. Proposition 1.3.2 below states the necessary and sufficient conditions for the special case under which SL_{tot} with Acceptable Waiting Time (AWT) of zero and single-server groups is jointly concave in λ_j and r_j.

Proposition 1.3.2. *When each group has exactly one server, and the acceptable waiting time is zero, then SL_{tot} is jointly concave in $\{\lambda_j\}$ and $\{r_j\}$ if and only if for each group $j \in J$, $\tau_{ij} = \tau_j \ \forall i \in FI_j$, i.e., when the mean service time for jobs routed to group j is independent of job type i.*

Proof. See Appendix B. □

The following proposition shows that SL_{tot} is not even pseudo-concave in its variables.

Proposition 1.3.3. *Even for the special case with one server per group with acceptable waiting time of zero, SL_{tot} is not jointly pseudo-concave in $\{\lambda_j\}$ and $\{r_j\}$.*

Proof. See Appendix C. □

Proposition 1.3.2 provides a special case for which SL_{tot} is concave, namely, when the service time is independent of the job type. But, Proposition 1.3.3 shows that in general the service level function SL_{tot} is not pseudo-concave. Notice that pseudo-concavity is a weaker condition that concavity; that is, SL_{tot} is neither concave nor pseudo-concave in the general case. Our numerical experiments in Section 1.5 show that KNITRO, a commercial nonlinear solver which can solve convex and pseudo-convex optimization problems to optimality, gets stuck in an infeasible point when we attempt to solve Problem (P) with the SL_{tot} objective. In the next section we introduce a tailored algorithm to solve this non-convex problem.

1.4 Fixed-ratio Shifting Envelopes and Shifting Quadratic Envelopes

We show that the Shifting Quadratic Envelopment (SQE) algorithm originally introduced in Cho *et al.* (2014) to solve a special case of our problem (i.e., when the acceptable waiting time is zero) can be extended to solve Problem (P) to for the general case where the acceptable waiting time is any non-negative number. In theory, the resulting convex math program can be made to approximate the original non-convex math program with arbitrary precision. We build on SQE and propose a more efficient solution algorithm, which we call Fixed-ratio Shifting Envelopes (FSE). The FSE algorithm exploits the structure of the service level function in Problem (P) and achieves better performance than SQE. We explain the difference between FSE and SQE in Section 1.4.3.

1.4.1 Overview

The key to the outer-approximation that underlies both SQE and FSE is to underestimate the mean service time that a job (regardless of the job type) experiences at each group, namely, $\tau_j = r_j/\lambda_j$. Fixing the mean service time $\tau_j = \tau_j^{FIX}$, we let $r_j(\lambda_j|\tau_j^{FIX}) = \tau_j^{FIX}\lambda_j$ denote the workload r_j solely as a function of λ_j. Then, we define the SL function with fixed τ_j's as

$$SL_{tot}^{FIX} := \sum_{j\in J} \lambda_j SL(\lambda_j, r_j(\lambda_j|\tau_j^{FIX})) = \sum_{j\in J} \lambda_j SL(\lambda_j, \tau_j^{FIX}\lambda_j), \tag{1.15}$$

where τ_j^{FIX} is any fixed mean service time less than or equal to the mean service time of the optimal solution, i.e., $\tau_j^* = r_j^*/\lambda_j^*$. Defining $\tau_j^{MIN} = \min_i \tau_{ij}$ and $\tau_j^{MAX} = \max_i \tau_{ij}$, we can

bound $r_j = \tau_j \lambda_j$ as follows,

$$\tau_j^{MIN} \lambda_j \leq r_j \leq \tau_j^{MAX} \lambda_j. \tag{1.16}$$

Since before solving problem (P) we know only that $\tau_j^* \in [\tau_j^{MIN}, \tau_j^{MAX}]$, the best choice for τ_j^{FIX} that is always guaranteed to underestimate the optimal mean service time is τ_j^{MIN}. Proposition 1.4.1 proves that SL_{tot}^{RL} defined as SL_{tot}^{FIX} with $\tau_j^{FIX} = \tau_j^{MIN}$ is a *relaxation* of SL_{tot}. That is, SL_{tot}^{RL} provides an upper bound for SL_{tot}, which we seek to maximize. Proposition 1.4.1 also proves that SL_{tot} is concave in $\{\lambda_j\}$, when $\{\tau_j\}$ is fixed.

Proposition 1.4.1. *Let $\tau_j^* = r_j^*/\lambda_j^*$ be the mean service time of the optimal solution (λ_j^*, r_j^*). For any fixed mean service time τ_j^{FIX} such that $\tau_j^{FIX} \leq \tau_j^*$, the function $SL_{tot}^{FIX} = \sum_{j \in J} \lambda_j SL(\lambda_j, \tau_j^{FIX} \lambda_j)$ is (i) separable by group, (ii) concave in λ_j for each group j, and (iii) an upper bound for $SL_{tot} = \sum_{j \in J} \lambda_j SL(\lambda_j, \tau_j^* \lambda_j)$.*

Proof. See Appendix D. □

Finally, since the constraints of the original problem (P) are convex, the following *relaxed problem* (RL) is a convex program,

$$\begin{aligned} (RL) \quad \max \quad & \sum_{j \in J} \lambda_j SL(\lambda_j, \tau_j^{MIN} \lambda_j) \\ \text{s.t.} \quad & (1.2), (1.3), (1.9), (1.10), (1.11), \text{and } (1.12). \end{aligned}$$

Every feasible solution in (RL) is also feasible in the original problem (P), because all constraints in (P) are present in (RL). To find a good solution for (P), we evaluate the solution found by (RL) by plugging it into the objective function of the original problem (P). The total optimality gap across all groups is the difference between the objective value

of (RL) and the value of this solution evaluated in the original objective,

$$\begin{aligned} Gap^{tot} :=& \sum_{j \in J} \lambda_j SL(\lambda_j, \tau_j^{MIN} \lambda_j) - \sum_{j \in J} \lambda_j SL(\lambda_j, r_j) \\ =& \sum_{j \in J} \lambda_j \Big[SL(\lambda_j, \tau_j^{MIN} \lambda_j) - SL(\lambda_j, r_j) \Big]. \end{aligned} \tag{1.17}$$

Since (i) $SL(\lambda, \tau\lambda)$ is decreasing in τ (see inequality (D.6) for Lemma D.1 in Appendix D), and (ii) $\tau_j^{MIN} \leq \tau_j$, then $SL(\lambda_j, \tau_j^{MIN}\lambda_j) \geq SL(\lambda_j, r_j)$ holds and the total optimality gap is always non-negative. The solution might be globally optimal even if the optimality gap is strictly positive. In other words, the optimality gap gives us a conservative (i.e., pessimistic) indicator of how close we are to a global optimum.

We may now be interested in how to improve the solution produced by (RL). One method, which we find fruitful and on which SQE and FSE is based, tightens the gap by introducing successively better underestimates τ_j^{FIX} in place of τ_j^{MIN} in (RL). We can achieve some further progress by also solving a subproblem (SP_r), in which we fix r_j found by (RL) and re-solve for the remaining variables, x_{ij} and λ_j (for details see Appendix E).

1.4.2 Shifting Algorithm

In Section 1.4.1 we explained how we find a feasible solution to the original problem (P) by solving the relaxed problem (RL). In this section, we explain how the SQE and FSE algorithms search for tighter relaxations (i.e., lower upper bounds for SL_{tot}) to close the optimality gap.

The contribution of group j to the objective function SL_{tot} depends on the arrival rate λ_j and the workload r_j (i.e., $\lambda_j SL(\lambda_j, r_j)$ is a function of variables λ_j and r_j). For each group j, the feasible region is a cone starting from the origin with the x-axis being the arrival rate λ_j and the y-axis being the workload r_j as depicted in Figure 1.3. The mean service

time $\tau_j = r_j/\lambda_j$ is the slope of the ray that starts from the origin and goes through the solution (λ_j, r_j). As defined in Section 1.4.1, let $\tau_j^* = r_j^*/\lambda_j^*$ be the mean service time of the optimal solution (λ_j^*, r_j^*). Our goal is to pick a mean service time τ_j^{FIX} that underestimates the optimal mean service time τ_j^* as closely as possible. This will allow us to overestimate the non-concave function $\lambda_j SL(\lambda_j, r_j)$ using the concave function $\lambda_j SL(\lambda_j, \tau_j^{FIX}\lambda_j)$ without introducing much approximation error. Since before solving problem (P) we know only that $\tau_j^* \in [\tau_j^{MIN}, \tau_j^{MAX}]$, the best choice for τ_j^{FIX} that is always guaranteed to underestimate the optimal mean service time is τ_j^{MIN}. This is why, in Section 1.4.1, we used the underestimate $\tau_j^{FIX} = \tau_j^{MIN}$ to construct (RL).

We call our algorithm Fixed-ratio Shifting Envelopes rather than Shifting Quadratic Envelopes developed by Cho *et al.* (2014) because the envelopes constituting the outer approximation in our general setting are not quadratic as they are in the special case when there is a single server at each group and the acceptable waiting time is zero as modeled by Cho *et al.* (2014). That is, when $AWT = 0$ and $k_j = 1$, the term $\lambda_j SL(\lambda_j, \tau_j^{FIX}\lambda_j)$ reduces to the quadratic term $\tau_j^{FIX}\lambda_j^2$, which was the focus of Cho *et al.* (2014).

We will now describe how we can use other estimates for τ_j^{FIX} that lead to tighter relaxations. As shown in Figure 1.4 we can subdivide the cone into m_j slices of arbitrary size by splitting the domain of τ_j into subdomains $[\tau_{j,1}, \tau_{j,2}], [\tau_{j,2}, \tau_{j,3}], \ldots, [\tau_{j,m_j}, \tau_{j,m_{j+1}}]$, where $\tau_{j,1} < \tau_{j,2} < \tau_{j,3} < \cdots < \tau_{j,m_j} < \tau_{j,m_{j+1}}$, $\tau_{j,1} = \tau_j^{MIN}$ and $\tau_{j,m_{j+1}} = \tau_j^{MAX}$. When the solution (λ_j, r_j) is in the n-th subdomain (i.e., in the slice defined by $\tau_{j,n}\lambda_j \leq r_j \leq \tau_{j,n+1}\lambda_j$), we can use $\tau_j^{FIX} = \tau_{j,n}$ as an underestimate for the mean service time τ_j at the point (λ_j, r_j). To keep track of which subdomain group j's mean service time is in, we use binary variables y_{jn} for $n \in N_j = \{1, 2, \ldots, m_j\}$, where y_{jn} is 1 if group j's mean service time is in the subdomain $[\tau_{j,n}, \tau_{j,n+1}]$, and is 0 otherwise. We use the constraint $\sum_{n \in N_j} y_{jn} = 1$ to restrict each group's mean service time to exactly one subdomain. Whenever the solution is in the n'th subdomain, we replace r_j with its corresponding underestimate, $r_{jn} = \tau_{j,n}\lambda_{jn}$. The full

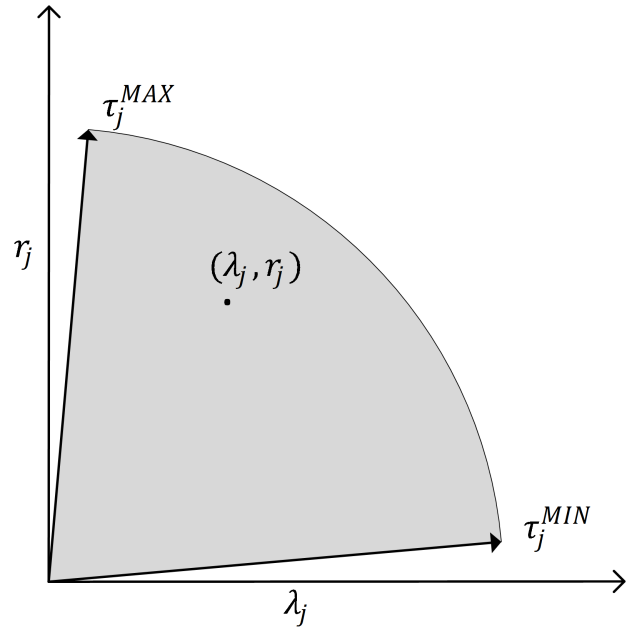


Figure 1.3: Illustration of a feasible point (λ_j, r_j) in the subspace for group j.

formulation of $(RL2)$ for the SQE-relaxed (or, equivalently FSE-relaxed) problem (RL) is as follows,

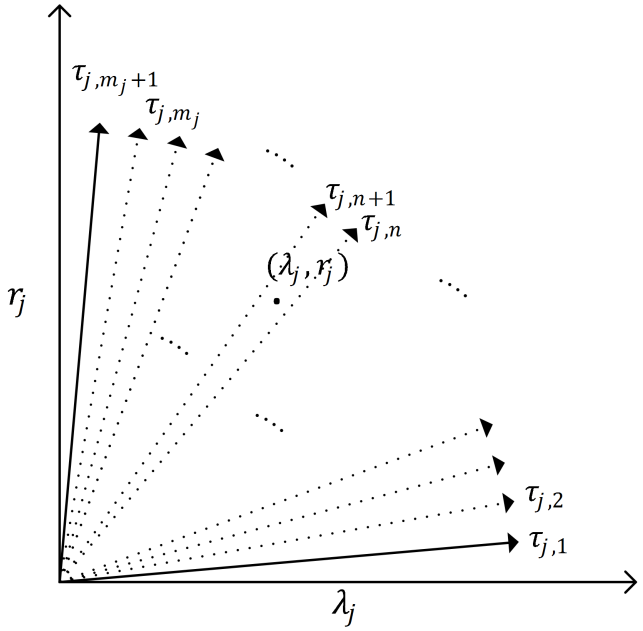


Figure 1.4: Illustration of the conic slices in SQE and FSE algorithms.

$$(RL2) \quad \max \quad \sum_{j\in J}\sum_{n\in N_j} \lambda_{jn} SL(\lambda_{jn}, \tau_{j,n}\lambda_{jn}) \tag{1.18}$$

$$\text{s.t.} \quad (1.2),\ (1.3),\ (1.9),\ (1.10),\ (1.11),\ (1.12),$$

$$\lambda_j = \sum_{n\in N_j} \lambda_{jn} \quad \forall j \in J, \tag{1.19}$$

$$r_j = \sum_{n\in N_j} r_{jn} \quad \forall j \in J, \tag{1.20}$$

$$\sum_{n\in N_j} y_{jn} = 1 \quad \forall j \in J, \tag{1.21}$$

$$0 \le \lambda_{jn} \le \lambda_j^{MAX} y_{jn} \quad \forall j \in J, \forall n \in N_j, \tag{1.22}$$

$$0 \le r_{jn} \le r_j^{MAX} y_{jn} \quad \forall j \in J, \forall n \in N_j, \tag{1.23}$$

$$\tau_{j,n}\lambda_{jn} \le r_{jn} \le \tau_{j,n+1}\lambda_{jn} \quad \forall j \in J, \forall n \in N_j, \tag{1.24}$$

$$y_{jn} \in \{0,1\} \quad \forall j \in J, \forall n \in N_j, \tag{1.25}$$

where λ_j^{MAX} and r_j^{MAX} are sufficiently big numbers. In addition to the variables and constraints of (RL), problem $(RL2)$ includes binary variables y_{jn} that define subdomain membership, continuous variables λ_{jn} and r_{jn}, and several logical constraints that link these quantities. Constraints (1.19) and (1.20) define the arrival rates λ_j and the workloads r_j for each group j, respectively. Constraint (1.21) ensures that only one conic slice is active. For inactive slices, i.e., when $y_{jn} = 0$, constraints (1.22) and (1.23) ensure that the corresponding arrival rate λ_{jn} and workload r_{jn} are zero, respectively. For active slices, i.e., when $y_{jn} = 1$, constraints (1.22) and (1.23) limit the arrival rate λ_{jn} and the workload r_j by their maximum values, i.e., λ_j^{MAX} and r_j^{MAX}, respectively. Constraint (1.24) connects r_{jn} to λ_{jn} and $\tau_{j,n}$. Similar to (RL), $(RL2)$ is also convex in the $\{\lambda_j\}$ variables.

As we divide the domain of each τ_j into finer intervals, our relaxation $(RL2)$ becomes tighter. Moreover, in the (λ_j, r_j)-space depicted in Figure 1.4, as the number of slices $m_j \to \infty\ \forall j$, the

area of each slice collapses to zero and the optimal value of $(RL2)$ converges to the optimal value of (P); that is, in theory we can approximate (P) to any arbitrary precision by simply slicing the subdomains of τ_j finely enough. However, as the number of slices increases, the problem $(RL2)$ becomes harder to solve because of the larger number of binary variables $\{y_{jn}\}$. Consequently, one would like to use an algorithm that uses a limited number of slices, defined at just the right values to approximate τ_j^* both quickly and accurately. The SQE and FSE algorithms begin with only one slice defined for each group; i.e., $m_j = 1, \tau_{j,1} = \tau_j^{MIN}$, and $\tau_{j,2} = \tau_j^{MAX}\ \forall j$. At each iteration we add new boundaries (or, equivalently new slices), and remove some boundaries as needed to keep the number of the boundaries and thus the computational expense of problem $(RL2)$ in check. The net result of adding and removing slice boundaries at each iteration can be thought of as *shifting boundaries*. Table 1.2 formally defines the *Shifting* algorithm common for both the *Fixed-ratio Shifting Envelopes* and the *Shifting Quadratic Envelopes* algorithms. In the following subsection we answer two very important questions, (i) *Adding boundaries:* Where should we add a new boundary? (ii) *Removing boundaries:* Which boundaries should we keep and which ones should we remove? As explained in the *Shifting* algorithm, both SQE and FSE add and remove boundaries. However, these algorithms differ in the way they add and remove boundaries. Most importantly, our FSE method exploits the specific functional form of the Service Level (SL) function to improve the performance of the shifting algorithm. Below, we explain in detail the adding and removing subroutines of the *Shifting* algorithm.

1.4.3 Adding and Removing Boundaries

In this section we explain *adding* and *removing* boundaries for SQE and FSE algorithms. For each group we start off by adding a new boundary to get closer to the optimal mean service time, τ_j^*. Next, we remove boundaries to keep the total number of binary variables (or, equivalently the total number of conic slices), i.e., y_{jn} in Problem $(RL2)$, in check.

Table 1.2: *Shifting* Algorithm (for SQE and FSE)

1. Required Parameters.

- tol = the optimality gap target; once we get below this threshold, we terminate.

2. Initialization.

- Set iteration counter $t \leftarrow 1$; lower bound $LB \leftarrow -\infty$; upper bound $UB \leftarrow \infty$; optimality gap $Gap^{tot} \leftarrow \infty$; and best solution found thus far $bestSol \leftarrow 0$.

- Initialize boundaries:

  $\tau_{j,1} \leftarrow \tau_j^{MIN}$; $\tau_{j,2} \leftarrow \tau_j^{MAX}$; and $T_{j,1} \leftarrow \{\tau_{j,1}, \tau_{j,2}\}$ $\forall j$.

3. Main Algorithm

- Repeat until the $Gap^{tot} \leq tol$

  • Solve the relaxed problem ($RL2$). Store the optimal value as z^{RL} and the optimal solution as Θ.

  • Evaluate solution Θ in the original problem objective function to obtain a feasible solution with the value of z^{EVAL}.

  • Update the best solution found thus far: If $z^{EVAL} > LB$ then, $bestSol \leftarrow \Theta$.

  • Update the lower bound: $LB \leftarrow \max(LB, z^{EVAL})$

  • Update the upper bound: $UB \leftarrow \min(UB, z^{RL})$

  • Update the optimality gap percentage: $Gap^{tot} \leftarrow (LB - UB)/UB$.

  • If $Gap^{tot} < tol$,

    ◦ Terminate and return $bestSol$, LB, UB, and Gap^{tot};

  else,

    ◦ **Add boundaries** (For FSE, see subroutine FSE^{ADD} in section 1.4.3 and Table F.1. For SQE, see section 1.4.3);

    ◦ **Remove boundaries** (For FSE, see subroutine FSE_{gr}^{REM} in section 1.4.3 and Table F.2. For SQE, see section 1.4.3).

  • Increment the iteration counter $t \leftarrow t + 1$, and repeat the loop.

Adding Boundaries

At each iteration of *Shifting*, we solve $(RL2)$ to get the (P)-feasible solution (λ_j, r_j), which lies in a particular slice, say the slice defined by $\tau_{j,n_j^{CURRENT}}\lambda_j \leq r_j \leq \tau_{j,n_j^{CURRENT}+1}\lambda_j$ for some $n_j^{CURRENT}$. We then subdivide this slice into two slices of arbitrary size, cutting it in two along the ray $r_j(\lambda_j|\tau_j^{NEW}) = \tau_j^{NEW}\lambda_j$, where $\tau_j^{NEW} = \theta\tau_{j,t}^{ACTUAL} + (1-\theta)\tau_{j,n_j^{CURRENT}}$ with $\theta \in [0,1]$ is the weighted average of $\tau_{j,n_j^{CURRENT}}$, i.e., the current underestimate of the mean service time at the point (λ_j, r_j), and the actual mean service time $\tau_{j,t}^{ACTUAL} = r_j/\lambda_j$ at the point (λ_j, r_j).

SQE: Cho *et al.* (2014) chose a fixed $\theta = 0.5$, which tended to work well for their problem instance.

FSE: We dynamically generate the new boundary between $\tau_{j,n_j^{CURRENT}}$ and $\tau_{j,t}^{ACTUAL}$ based on the difference between the value of the service level evaluated at the current solution (λ_j, r_j), and the value of the service level evaluated at the lower boundary point $(\lambda_j, \tau_{j,n_j^{CURRENT}}\lambda_j)$, which is immediately below (λ_j, r_j). Note that if we generate the new boundary closer to the current solution (i.e., choosing θ close to 1), we increase the cost of the current solution in the objective of $(RL2)$ such that in the following iteration we are less likely to generate solutions in the new slice, above τ_j^{NEW}. Conversely, since the current solution is not necessarily the optimal solution to the original problem (P) and can change in the following iteration, we may not want to spend many iterations only making small changes to the lower boundary (i.e., choosing θ near 0).

As is shown in Figure 1.2, for ranges with relatively low workload (e.g., for $r \leq 1$ which in this example with $k = 5$ servers corresponds to utilization below 20%), the Erlang-C function is relatively insensitive to changes in workload, but as workload increases, Erlang-C becomes progressively more sensitive to changes in workload (i.e., its slope increases). In turn, since the Service Level (SL) is a function of Erlang-C, it also exhibits this property. In addition to

the SL function's sensitivity to workload, it is also sensitive to the acceptable waiting time, T, which is present in the exponent of the SL function, i.e., $e^{-\left(\frac{\lambda(k-r)}{r}\right)T}$ in (1.6). In fact, for $T = 5$ and 20, the SL function is more sensitive to workload compared to when $T = 0$ (i.e., when SL reduces to Erlang-C). These two observations, namely, sensitivity to workload and acceptable waiting time, motivated our development of the FSE algorithm for Problem (P).

For small values of workload we can update the slice boundary more aggressively (i.e., set θ closer to 1) to move quickly toward the current solution, knowing that the move does not appreciably increase the cost of the boundary in the objective of $(RL2)$. Conversely for large values of workload, we move gradually toward the current solution (i.e., set θ closer to 0), since the impact of the move on the objective function is significant and can make the new boundary much more costly in the objective function.

For each group j, we compute SL evaluated at the current lower boundary, i.e., $SL(\lambda_j, \tau_{j,n_j^{CURRENT}}\lambda_j)$, and the SL evaluated at the actual mean service time (λ_j, r_j), using the value of $\tau_{j,t}^{ACTUAL} = \lambda_j/r_j$, i.e., $SL(\lambda_j, \tau_{j,t}^{ACTUAL}\lambda_j)$. Then, we define the normalized difference between the Service Level evaluations as,

$$\delta = \frac{SL(\lambda_j, \tau_{j,n_j^{CURRENT}}\lambda_j) - SL(\lambda_j, \tau_{j,t}^{ACTUAL}\lambda_j)}{SL(\lambda_j, \tau_j^{MIN}\lambda_j) - SL(\lambda_j, \tau_j^{MAX}\lambda_j)}. \tag{1.26}$$

When δ is large, we generate the new boundary closer to the current lower boundary through choosing smaller values for θ; conversely when δ is small, we move rapidly toward the current solution by choosing larger values for θ. Also, suggested by our experiments, we keep θ away from the extreme values of zero and one by enforcing $\theta^{MIN} \leq \theta \leq \theta^{MAX}$. Our formula for θ is

$$\theta = (1 - \delta)(\theta^{MAX} - \theta^{MIN}) + \theta^{MIN}. \tag{1.27}$$

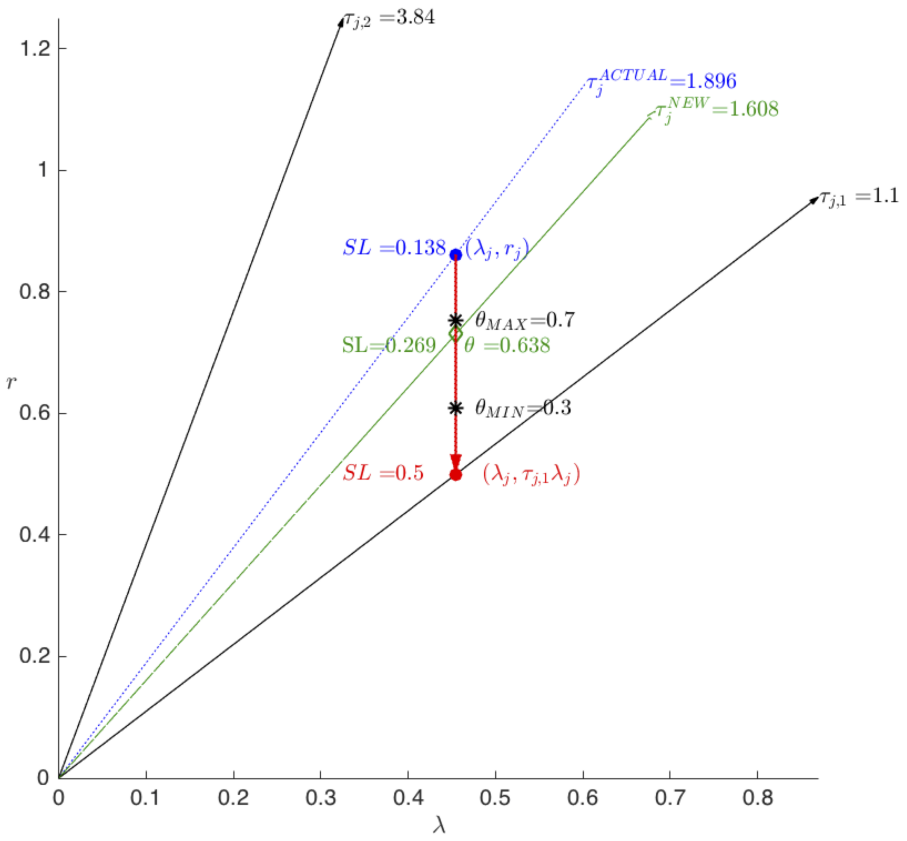


Figure 1.5: Choosing θ according to the difference between the Service Level evaluated at the actual mean service time and the Service Level evaluated at the lower slice boundary.

We choose $\theta^{MIN} = 0.3$ and $\theta^{MAX} = 0.7$ in our FSE algorithm. Figure 1.5 illustrates how we generate a new boundary according to equations (1.26) and (1.27) with acceptable waiting time of zero for a group with 3 servers. We denote the set of boundaries at iteration t by $T_{j,t}$ and the set of boundaries after adding a new boundary, i.e., the output of the *adding boundaries* subroutine, FSE^{ADD}, by $T_{j,t}^{add}$. Table F.1 in Appendix F formally defines the *adding boundaries* subroutine, FSE^{ADD}, for the FSE algorithm.

Removing Boundaries

Knowing that the complexity of $(RL2)$ increases by the total number of binary variables y_{jn}, i.e., the total number of slices, we limit the total number of slices that we keep. The algo-

rithms SQE and FSE use different approaches to decide which boundaries to keep/remove.

SQE: Motivated by the desire to underestimate the optimal τ_j, Cho *et al.* (2014) keep three boundaries below and one slice boundary above the actual boundary at iteration t, $\tau_{j,t}^{ACTUAL}$. However, a boundary that is below the current solution $\tau_{j,t}^{ACTUAL}$ produced in this iteration may be above the $\tau_{j,t}^{ACTUAL}$ in following iterations. In addition, we may decide to allocate more boundaries to one group as opposed to others, since not all groups have the same impact on the total gap defined in (1.17). FSE addresses both these aspects.

FSE: As opposed to SQE that keeps four boundaries for each group (i.e., one above and three below), FSE maintains a total of ψ^{tot} boundaries across all groups, with no group being allocated more than ψ^{gr} boundaries. We notice that ψ^{tot} is closely associated to the computational complexity of the problem, i.e., the total number of binary variables, y_{jn}'s. To remove boundaries from groups, FSE addresses two questions, separately: (i) how many boundaries do we remove from each group (subroutine FSE_{gr}^{REM})?; and (ii) for the group(s) that were chosen to lose boundaries, which boundaries do we remove (subroutine FSE_{bnd}^{REM})? Notice that FSE_{bnd}^{REM} runs for each group, separately.

FSE_{gr}^{REM}: For each group that ψ^{gr} is reached, we remove one boundary from that group to maintain ψ^{gr}. One the other hand, if ψ^{tot} is reached, we remove boundaries from groups that have less potential for closing the total gap, Gap^{tot}, defined in (1.17). We compute the contribution of each group to the total gap,

$$Gap_j := \lambda_j \Big[SL(\lambda_j, \tau_{j,n_j^{CURRENT}} \lambda_j) - SL(\lambda_j, r_j) \Big] \quad \forall j \in J, \tag{1.28}$$

where $\tau_{j,n_j^{CURRENT}}$ is the current underestimate for τ_j and $Gap^{tot} = \sum_{j \in J} Gap_j$. Note that since $SL(\lambda_j, \tau_{j,n_j^{CURRENT}} \lambda_j) \geq SL(\lambda_j, r_j)$ then Gap_j is non-negative. In allocating boundary budget across groups, we give groups with larger Gap_j a higher priority.

In an ordered list called $Sort$, we sort groups from smallest to largest Gap_j. We traverse this list in order and remove boundaries from the groups to meet ψ^{tot}. We pick the group at the top of the list, j_{top}, and remove it from $Sort$. If j_{top} has five or more boundaries (i.e., τ_j^{MIN} and τ_j^{MAX}, and at least 3 boundaries in between), we remove two boundaries and if j_{top} has exactly four boundaries, then we remove one boundary (see FSE_{bnd}^{REM} explained below). If j_{top} has three boundaries we do not remove any boundary. We pick the next group from the top of $Sort$ and continue. Table F.2 in Appendix F formally defines subroutine FSE_{gr}^{REM} which identifies boundaries to be removed from a specific group.

FSE_{bnd}^{REM}: The main inputs to subroutine FSE_{bnd}^{REM} are (i) $ToRemove$: the number of boundaries to be removed from a specific group (see the output of FSE_{gr}^{REM} in Table F.2 of Appendix F), and (ii) $T_{j,t}^{add}$: the ordered list of boundaries with the newly generated boundary for iteration t (see the output of subroutine FSE^{ADD} in Table F.1 of Appendix F). The subroutine FSE_{bnd}^{REM} works as follows. We formally define FSE_{bnd}^{REM} in Table F.3 of Appendix F.

For a group j that we want to remove a boundary from, we keep each existing boundary proportional to the likelihood that τ_j falls in the corresponding slice. We use the actual mean service times, $\tau_{j,t}^{ACTUAL}$'s corresponding to the solution produced at the current iteration t, and denote the set $\{A_{j,t}\} = \{\tau_{j,1}^{ACTUAL}, \ldots, \tau_{j,t}^{ACTUAL}\}$ as the set of actual mean service times from iteration 1 to t, the current iteration. For each boundary $\tau_{j,n} \in T_{j,t}^{add} \setminus \tau_j^{MAX}$, we calculate its score, i.e., the number of actual solutions that are in the slice boundary defined by $\tau_{j,n}$ (i.e., that fall above $\tau_{j,n}$ and below $\tau_{j,n+1}$),

$$S(\tau_{j,n}) = \sum_{\tau \in A_{j,t}} I(\tau_{j,n} \leq \tau < \tau_{j,n+1}) \quad \forall \tau_{j,n} \in T_{j,t-1}^{add} \setminus \tau_j^{MAX}, \tag{1.29}$$

where $I(.)$ is an indicator function. Note that we do not define a score for τ_j^{MAX}, since it does not underestimate any actual mean service time. The higher the score of a boundary,

the higher the chance that we keep that boundary. We always keep τ_j^{MIN} and τ_j^{MAX} to cover all service times as well as τ_j^{NEW}. After calculating the scores we produce the set of boundaries that are eligible for removal, $T_j^{temp} \leftarrow T_{j,t}^{add} \setminus \{\tau_j^{MIN}, \tau_j^{NEW}, \tau_j^{MAX}\}$. Next, we form a subset of T_j^{temp} for boundaries with a score of zero, $T_j^{zero} \leftarrow \{\tau_{j,n} \in T_j^{temp} | S(\tau_{j,n}) = 0\}$. If there are boundaries with a score of zero, we remove those boundaries first, removing as many boundaries as we have decided to remove for this group, breaking ties randomly if the number of boundaries with a score of zero exceeds the number of boundaries we must remove. We drop the removed boundaries from the set T_j^{temp}. If at this point there is no boundary with a score of zero left in T_j^{temp} and we need to remove more boundaries, we keep boundaries probabilistically in proportion to their scores. We calculate the cumulative score for each boundary left in the set T_j^{temp} as,

$$S^{cum}(\tau_{j,n}) \leftarrow \sum_{\tau \in T_j^{temp}: \tau \leq \tau_{j,n}} S(\tau) \quad \forall \tau_{j,n} \in T_j^{temp}. \tag{1.30}$$

We generate a random number, $rand$, from the following uniform distribution with support spanned by zero and the cumulative score of the highest boundary in T_j^{temp},

$$U[0, S^{cum}(\tau_{j,|T_j^{temp}|})]. \tag{1.31}$$

We select the boundary $\tau_{j,n} \in T_j^{temp}$, where $S^{cum}(\tau_{j,n-1}) \leq rand < S^{cum}(\tau_{j,n})$ and keep the selected boundary by transferring it from T_j^{temp} to a new set $T_j^{keep} = \{\tau_j^{MIN}, \tau_j^{NEW}, \tau_j^{MAX}\}$ (i.e., the set of boundaries that we keep),

$$T_j^{temp} \leftarrow \{T_j^{temp}\} \setminus \tau_{j,n} \text{ and } T_j^{keep} \leftarrow \{T_j^{keep}\} + \tau_{j,n}. \tag{1.32}$$

While there are still more boundaries to be removed, we repeat this loop by updating bound-

ary cumulative scores according to (1.30), generating new random numbers according to (1.31), and transferring boundaries from T_j^{temp} to T_j^{keep} according to (1.32). The subroutine FSE_{bnd}^{REM} terminates by keeping the boundaries left in T_j^{keep} as well as τ_j^{MIN}, τ_j^{NEW}, and τ_j^{MAX} in the set of boundaries kept after removal,

$$T_{j,t+1} \leftarrow T_j^{keep}.$$

In the following section we illustrate the progression of the FSE algorithm on a small example.

1.4.4 FSE: an illustrative example

Figure 1.6 illustrates the progression of the FSE algorithm on an example with 2 job types and 2 groups. We keep at most a total of 8 boundaries, i.e., $\psi^{tot} = 8$, with a limit of 5 boundaries for each group, i.e., $\psi^{gr} = 5$.

At iteration 1, for each group we begin with a single slice bounded by $\tau_{j,1}$ and $\tau_{j,2}$, where $\tau_{j,1} = \tau_j^{MIN}$ and $\tau_{j,2} = \tau_j^{MAX}$. Assuming that we have not reached the target optimality gap we apply FSE^{ADD} to add a new boundary between $\tau_{j,1}$ and $\tau_{j,2}$ (see Table F.1 in Appendix F). We solve $(RL2)$ and (SP_r), and use the current iteration solution (λ_j, r_j) to compute the actual mean service time $\tau_{j,1}^{ACTUAL} = r_j/\lambda_j$ ($\tau_{1,1}^{ACTUAL} = 3.14$ and $\tau_{2,1}^{ACTUAL} = 1.64$). The actual mean service time, $\tau_{j,1}^{ACTUAL}$, is the slope of the dotted blue ray in Figure 1.6. For each group, we split the subdomain of $[\tau_{j,1}, \tau_{j,2}]$ into two slices by introducing a new slice boundary τ_j^{NEW}, where $\tau_j^{NEW} = \theta\tau_{j,1}^{ACTUAL} + (1-\theta)\tau_{j,1}$. To simplify calculations, we set θ_{MIN} to 0 and θ_{MAX} to 1, which simplifies (1.27) to $\theta = 1 - \delta$. Using equations (1.26) and $\theta = 1 - \delta$ we calculate δ and θ; for both groups, δ and θ are 0.2 and 0.8, respectively. The new boundary with slope τ_j^{NEW} is depicted by the green dashed line in Figure 1.6. Now we have a total of 6 boundaries, 3 for each group, which we use at iteration 2. Since we have not

reached either ψ^{tot} or ψ^{gr}, we do not remove any boundary. After relabeling slice boundaries τ_j^{NEW} and $\tau_{j,2}$ as $\tau_{j,2}$ and $\tau_{j,3}$, respectively, we proceed to iteration 2.

At iteration 2 we apply FSE^{ADD} to add new boundaries. We re-solve [1] $(RL2)$ and subproblem (SP_r) to get a new point (λ_j, r_j) and its associated actual mean service time $\tau_{j,2}^{ACTUAL} = r_j/\lambda_j$. Since for both groups the points (λ_j, r_j) are in the bottom-most slices, we proceed by splitting the bottom-most slices into two. For each group, we do this by creating a new slice boundary with slope $\tau_j^{NEW} = \theta\tau_{j,2}^{ACTUAL} + (1-\theta)\tau_{j,1}$, where equation (1.27) determines θ. At the end of iteration 2 there are a total of 8 boundaries (4 boundaries for each group). We proceed to iteration 3 with these boundaries (see iteration 2 in Figure 1.6).

After applying FSE^{ADD} at iteration 3, although we do not exceed $\psi^{gr} = 5$ for any group, we now have reached a total of 10 boundaries (5 boundaries for each group), i.e., 2 more than our limit of $\psi^{tot} = 8$. To remove 2 boundaries before starting the following iteration, i.e., iteration 4, we apply FSE_{gr}^{REM} (see Table F.2 in Appendix F). We sort groups according to their gaps, Gap_j. Since $Gap_2 < Gap_1$, we remove 2 boundaries from group 2 without removing any boundary from group 1. To select 2 boundaries from group 2, we proceed as follows. Using equation (1.29) we calculate the scores for group 2's boundaries (The black circles in Figure 1.6 are (λ_j, r_j) points on which $\tau_{j,t}^{ACTUAL}$'s are based),

$$S(\tau_{2,1} = 1) = 0,\ S(\tau_{2,2} = 1.51) = 0,\ S(\tau_{2,3} = 1.56) = 2,\ \text{and}\ S(\tau_{2,4} = 1.65) = 1.$$

Since we always keep the newly generated and the bottom-most boundaries, we don't consider $\tau_2^{MIN} = \tau_{2,1}$ or $\tau_2^{NEW} = \tau_{2,3}$ for removal. Thus, the set of candidate boundaries for removing is $T_2^{temp} = \{1.51, 1.65\}$. We remove $\tau = 1.51$ since its score is zero. Because we should remove two boundaries, we also remove the only boundary left in $T_2^{temp} = \{1.65\}$. The 2 boundaries that are removed are depicted as red rays in Figure 1.6. Iteration 4 does not use

[1] Upon solving the relaxed problem $(RL2)$ and finding $\{x_{ij}\}$, $\{r_j\}$ and $\{\lambda_j\}$, we can seek a better solution by solving a *subproblem* (SP_r) (Check Appendix E for more details).

these two removed boundaries.

After applying FSE^{ADD} at iteration 4, group 1 has 6 boundaries and group 2 has 4 boundaries. We remove one boundary from group 1, to satisfy the $\psi^{gr} = 5$ limit. Also, to meet the $\psi^{tot} = 8$ limit we need to remove an additional boundary from either group 1 or 2 based on their contribution to the total optimality gap.

To remove a boundary from group 1, we calculate the scores for its boundaries,

$$S(\tau_{1,1} = 0.5) = 0,\ S(\tau_{1,2} = 1.3) = 1,\ S(\tau_{1,3} = 1.64) = 1,\ S(\tau_{1,4} = 1.96) = 1,\ \text{and}$$
$$S(\tau_{1,5} = 2.61) = 1.$$

Except for $\tau_1^{MIN} = \tau_{1,1}$, which we cannot remove, none of the scores are zero. Also, we cannot remove the new boundary $\tau_1^{NEW} = \tau_{1,3}$. After calculating cumulative scores (see equation (1.30)) and generating a random number for boundaries in $T_1^{temp} = \{1.3, 1.96, 2.61\}$ (see equation (1.31)) we remove $\tau_{1,5} = 2.61$. Now we have a total of 9 boundaries and to meet $\psi^{tot} = 8$, we need to remove another boundary. Since $Gap_2 < Gap_1$ we remove a boundary from group 2. The scores for group 2 boundaries are,

$$S(\tau_{2,1} = 1) = 0,\ S(\tau_{2,2} = 1.56) = 3 \text{ and } S(\tau_{2,3} = 2.39) = 1.$$

Except for $\tau_2^{MIN} = \tau_{2,1}$, which we are not allowed to remove, none of the scores are zero. Also, we cannot remove the new boundary $\tau_2^{NEW} = \tau_{2,3}$. We therefore remove the only boundary in T_1^{temp}, which is $\tau_{2,2} = 1.56$.

After applying FSE^{ADD} at iteration 5, group 1 has 6 boundaries, i.e., one more than $\psi^{gr} = 5$. We remove one of its boundaries. Then, we reach a total of 9 boundaries, 1 more than $\psi^{tot} = 8$. Since $Gap_1 < Gap_2$, we remove another boundary from group 1 boundaries and we meet $\psi^{tot} = 8$. Our FSE algorithm repeats these steps until we reach the desired optimality

gap.

1.5 Experiments

Note that although theoretically convergence is not guaranteed unless we continue to add boundaries at each iteration without removing any, our experimental results show that FSE is indeed able to find near-optimal solutions, and it does so faster than alternative benchmarks. We now run a number of numerical experiments to evaluate the performance of our FSE algorithm. In our instances, the acceptable waiting time is set at $T = 10$ time units.

1.5.1 Solvers for Problem Components

To run the FSE and SQE algorithms on our instances we need to invoke appropriate solvers to solve the relaxed problem ($RL2$), a mixed-integer problem, and to solve the subproblem (SP_r), a nonlinear convex problem. We deploy KNITRO, a powerful solver for solving convex optimization problems (Byrd *et al.*, 2006), to solve ($RL2$) and (SP_r) to optimality.

1.5.2 Benchmarks

We benchmark the performance of FSE against the following commercial solvers.

- **KNITRO:** Recall that in Section 1.3.3 we showed that the original problem (P) is non-convex. Since it can happen that a local nonlinear solver may still converge to a local optimum which is quite good or even optimal, we run KNITRO 10.1 on (P) as a benchmark.
- **BARON:** BARON (Tawarmalani & Sahinidis, 2005; Sahinidis, 2014) is a global solver

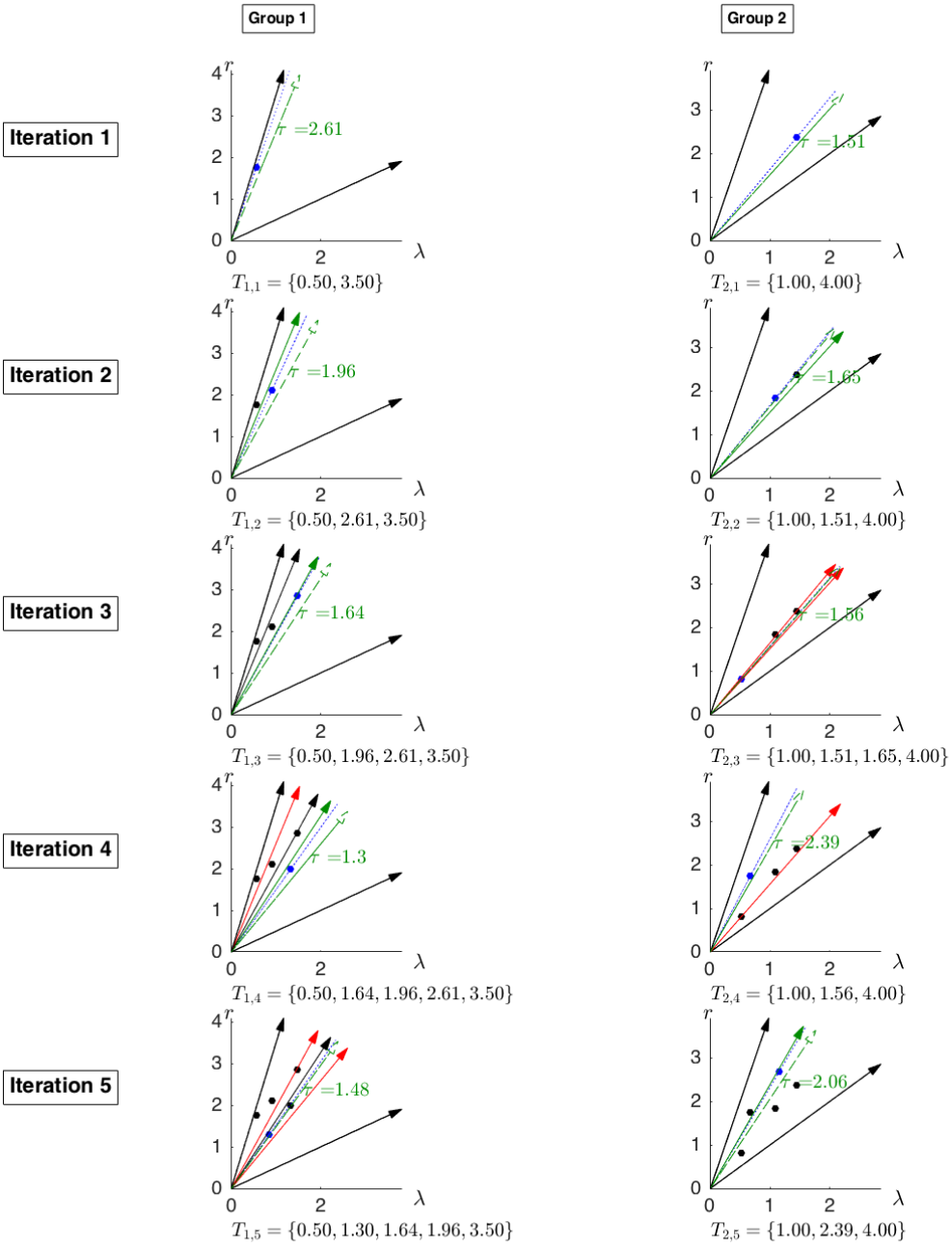


Figure 1.6: Illustration of our FSE algorithm.
$T_{j,t}$: ordered list of boundaries of group j at iteration t; *dotted blue arrow*: $\tau_j^{ACTUAL} = r_j/\lambda_j$ for the current iteration solution (λ_j, r_j); *dashed green arrow*: newly generated boundary to be used in the following iteration; *solid green arrow*: new boundary used in this iteration and generated in the previous iteration (In the previous iteration the green solid arrow was dashed to show that it was not used.); *solid red arrow*: boundary removed for the following iteration.

for solving general non-convex optimization problems. We use BARON 17.01.02 to find the global optimum for the original problem (P). We compare FSE with BARON in terms of solution quality and speed.

Alongside the above solvers, we compare FSE with the following algorithms.

- **Shifting Quadratic Envelopes (SQE):** We used the SQE algorithm in Cho *et al.* (2014) to solve the service level problem (see Section 1.4 for details).
- **Fixed-ratio envelopes with evenly-spaced boundaries (FE-even):** In this algorithm, we generate boundaries that are evenly-spaced between τ_j^{MIN} and τ_j^{MAX}. The algorithm runs for one iteration and does not shift boundaries.
- **Fixed-ratio envelopes with boundary adding (FE-add):** In this algorithm, we add boundaries iteratively and we stop when we reach the maximum number of boundaries. FE-add is FSE without removing any boundaries. By comparing FE-add with FSE, we test the performance of the boundary *shifting* part of FSE.

In Section 1.5.3 we explain how we generate our test instances. We discuss results in Section 1.5.4.

1.5.3 Data

Since FSE has different applications, we generate two sets of instances, (i) a *planar* dataset for transportation-on-demand systems and (ii) a *non-planar* dataset for call centers and load balancing systems.

For our planar instances, we generate random points for job nodes and group nodes on a two-dimensional plane. We define a coverage radius for each group to determine if a group

can serve a job node. If job i is covered by group j, i.e., $i \in FI_j$, the Euclidean distance between i and j gives us the service time, τ_{ij}.

For our non-planar instances, we generate τ_{ij}'s randomly from a uniform distribution with support on the interval $[0.5, 3.5]$. For each (i, j) combination, we generate a uniform random number between zero and one; if it's greater than a threshold we keep the link, otherwise we remove it. After removing links we form the adjacency sets FJ_i and FI_j.

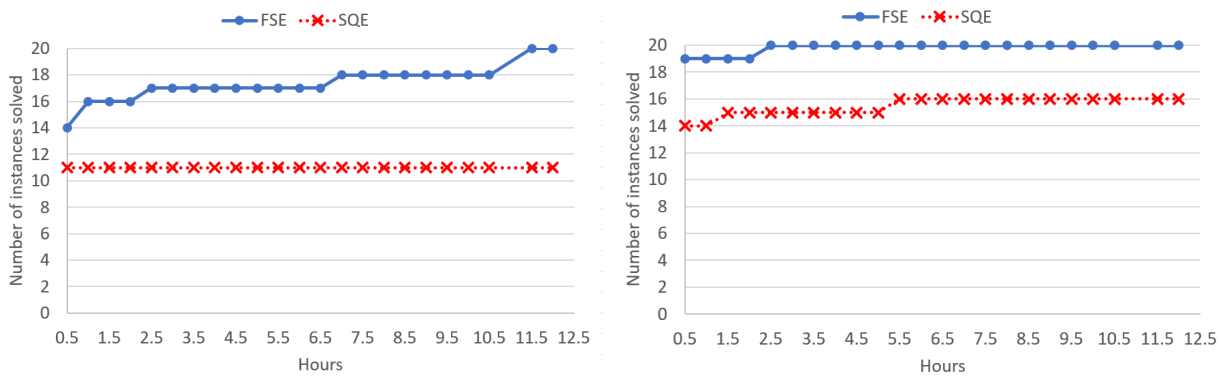
For both datasets, we generate demands d_i of each type i and number of servers k_j at each group j randomly according to the uniform distribution between 1 and 10 and the discrete uniform distribution between 1 and 10, respectively.

1.5.4 Results

We coded the algorithms in AMPL 2017.01.26 and used a Windows PC with 16 GB of RAM and a quad core CPU with 3.1 GHz. We ran algorithms for a maximum of 12 hours and only allowed our solvers and algorithms to go over 12 hours to finish running the current iteration. We terminated our instances when they reached a percentage gap of 1% (or 5%) defined as,

$$Gap^{pct} := 100 \times \frac{UB - LB}{UB}, \tag{1.33}$$

where LB and UB are the objective lower bound and upper bound, respectively. Note that the lower bound (LB) is also the best feasible solution since we are maximizing SL_{tot}. In Tables G.4 and G.6 in Appendix G, we list our results for solvers KNITRO and BARON, as well as solution algorithms FSE, SQE, FE-add, and FE-even. FSE outperforms all benchmarks by solving all twenty instances in the allotted time followed by SQE that solves thirteen instances to within $Gap^{pct} = 1$ and sixteen instances to within $Gap^{pct} = 5$. FE-add solves



(a) Target percentage gap of 1%

(b) Target percentage gap of 5%

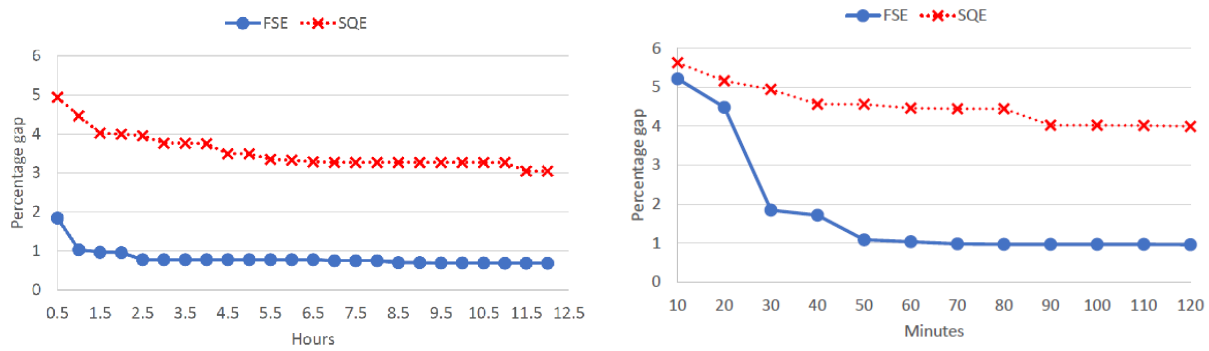
Figure 1.7: Number of instances (out of 20) solved to within the target percentage gap of 1% or 5%

eight instances to within $Gap^{pct} = 1$ and thirteen instances to within $Gap^{pct} = 5$, showing that *shifting boundaries* is necessary for FSE to close the gap. The results for FE-even show that evenly distributing the boundaries does not perform well. KNITRO gets stuck in a local infeasible point for all instances. Finally, BARON converges slowly to a global optimum and cannot close the gap in the allotted time for any of the 20 instances.

In Figure 1.7 we compare the performance of the FSE and SQE algorithms. As seen in both Figures 1.7a and 1.7b, FSE solves more instances than SQE for every allotted solution time budget, from 0.5 hour to 12.5 hours.

In Figure 1.8 we show how FSE and SQE close the gap over time. FSE quickly improves the bounds over the first 2 hours and reaches the target percentage gap of 1% while for SQE it takes longer to improve the bounds.

Our experiments show that an instance's complexity (i.e., the solution time) increases by the number of groups, $|J|$, which can be attributed to the number of binary variables y_{jn}'s in problem $(RL2)$ (see Figure 1.9). However, this observation has exceptions. For example, it takes less time for both FSE and SQE to solve instance 19 which has $|J| = 8$ server groups than to solve instance 18 which has $|J| = 7$ server groups. We observe that an instance's



(a) Over the allotted time of 12 hours

(b) Over the first 2 hours

Figure 1.8: Average percentage gap of 20 instances over time for FSE and SQE algorithms (Target percentage gap is 1%).

complexity does not depend on the instance type (i.e., planar or non-planar) or instance density (i.e., number of links $|F|/(|I| \times |J|)$).

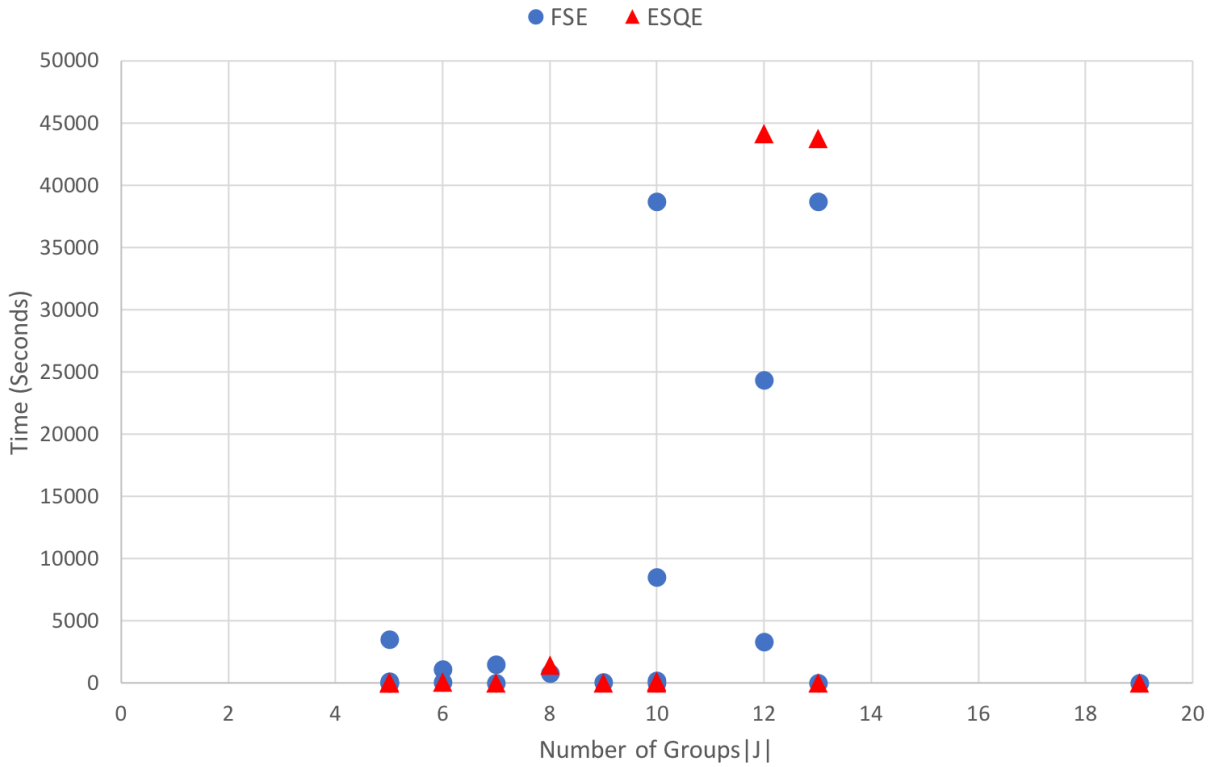


Figure 1.9: Instance complexity (solution time) vs. number of groups ($|J|$)

1.6 Extensions

Problem (P) has the Service Level in its objective. However, we may want to consider planning models which instead have the SL in a constraint. For example, in $(P2)$ below we show the case where we seek to maximize the number of jobs covered and there is a constraint on the service level that the jobs experience. For example, one might want to maximize the number of covered jobs such that the average service level is above 80%.

$$\begin{aligned} (P2) \quad \max \quad & \sum_{j \in J} \lambda_j \\ \text{s.t.} \quad & \text{Constraints (1.2), (1.3), (1.9), (1.10), (1.11), (1.12),} \\ & SL_{avg} \geq \alpha_{avg}^{SL}. \end{aligned} \tag{1.34}$$

FSE can be employed to deal with the inherent non-convexity of the service level constraint. We reason as follows. Assuming that the coverage $\sum_{j \in J} \lambda_j$ is strictly positive, according to the definitions of SL_{avg} and SL_{tot} in equations (1.8) and (1.7) constraint (1.34) can be written as,

$$SL_{tot} \geq \alpha_{avg}^{SL} \sum_{j \in J} \lambda_j. \tag{1.35}$$

Proposition 1.3.2 proves that the left-hand-side of constraint (1.35) is not jointly concave in λ_j and r_j. However, Proposition 1.4.1 proves that the function SL_{tot} is concave in λ_j when τ_j is fixed. Thus, constraint (1.35) is convex in λ_j with fixed τ_j. To apply FSE to these constraints, we notice that the relaxations of constraint (1.35) do not necessarily satisfy the corresponding original constraint, i.e., since the relaxed total service level SL_{tot}^{RL} is greater than or equal to the actual service level SL_{tot}, satisfying the relaxed constraint $SL_{tot}^{RL} \geq \alpha_{avg}^{SL} \sum_{j \in J} \lambda_j$ does not guarantee satisfying the original constraint $SL_{tot} \geq \alpha_{avg}^{SL} \sum_{j \in J} \lambda_j$. To overcome this, we must additionally solve a new *restricted problem* $(RS3)$ to find a feasible solution (this is

a step not needed when the service level is in the objective). In ($RS3$), we overestimate the workload by a boundary above the actual boundary. At each iteration, ($RS3$) yields a feasible solution which is a lower bound for the coverage objective function since ($P2$) is a maximization problem. To find an upper bound and the optimality gap we solve the *relaxed problem* ($RL3$), which uses boundaries below the actual boundary just like we saw with ($RL2$). Since problems ($RS3$) and ($RL3$) are solved independently we can apply different computational budgets for solving them if one is more critical than another for closing the gap for ($P2$). In our experience, we have found it beneficial to disproportionately spend more computational effort solving ($RS3$). For example, we can solve ($RL3$) with maximum number of three boundaries for a few iterations and solve ($RS3$) with maximum number of ten boundaries for more iterations. For further details, as well as the math programs for ($RL3$) and ($RS3$), see Appendix H.

1.7 Conclusions

Service level is an important performance measure for transportation networks and call centers. We showed that for multi-class multi-server systems, this performance measure is non-convex in the arrival rate and workload. Nevertheless, our experiments showed that our method, Fixed-ratio Shifting Envelopes (FSE), can efficiently solve the non-convex math program that plans the volume of flows from each job type to each server group. Our FSE method outperforms a related yet less specialized method called Shifting Quadratic Envelopes (SQE), which was originally developed by Cho *et al.* (2014) to solve the special case of our problem where acceptable waiting time is zero and each group has only one server. In contrast, we tailored our FSE method to accommodate (1) a more general performance measure, namely service level, and (2) multiple servers at each group. Our numerical experiments evaluate the performance of our FSE method for solving instances of this more

general problem class, and show that in addition to outperforming the SQE method, FSE also outperforms the commercial solvers KNITRO and BARON. Finally, we also tested simpler versions of FSE to understand how slice boundary selection and slice deletion impacts the performance of the algorithm. Our experiments confirm that FSE outperforms its simpler versions.

We also showed that the service level performance measure can appear in constraints as well as the objective function; for example, a limit for the average service level of a random job. One can use FSE to solve problems that have service level constraints. However, in this case the relaxed problem does not necessarily produce a feasible solution and we should overestimate the workloads by using the upper boundaries for mean service times and solving a restricted problem as well.

Our math programming model is useful for planning purposes and can be embedded in larger optimization problems such as location-allocation and call center staffing problems. It is also important to note that although our math program encodes a static allocation policy other papers have shown how to construct dynamic routing policies using the solution from (P) (see for example Nourbakhsh & Turner (2018), where we have developed "dynamized" routing policies for the (different) *expected waiting time* performance measure). In the same manner, we can use the solution of problem (P) (or, relaxed problems (RL) or $(RL2)$ to design routing policies for online settings.

Chapter 2

Dynamized Routing Policies for Minimizing Expected Waiting Time in a Multi-Class Multi-Server System

2.1 Introduction

We study the problem of minimizing expected waiting times in a multi-class multi-server queueing system, where the service times are both job-type and server-type dependent. For this queueing system the service time that a job experiences depends on the routing decision of the server to which the job is routed to. This endogeneity of the service time, i.e. the dependency of the service time on the routing decision, complicates the routing problem. In fact, the optimal dynamic policy for this queueing system is unknown (Mehrotra *et al.*, 2012). On the other hand, static policies derived from the solution of math programs have the added benefit of matching supply and demand in a richer way than simple dynamic policies, and in some cases such static policies can be "dynamized", i.e., combined with some elements of a

simpler dynamic policy to produce excellent results. We introduce a convex math program which we use to produce an optimal static policy, and then show how this static policy can be "dynamized" into a number of related dynamic policies which depend on the math program's solution. Using a comprehensive set of simulated instances, we then evaluate the performance of our dynamized policies. In our context, the *Fastest-Server-First* (FSF) policy is a dynamic policy that is known to be near-optimal for the single job-type special case (i.e., it is provably optimal in the Halfin-Whitt regime). We "dynamize" our static policy using FSF as our guide, and empirically show that several of our "dynamized" policies outperform FSF.

Specifically, we consider a routing problem for a multi-class multi-server system with Poisson arrivals and service rates that depend on both the job type and the server type. Each group of servers has a single queue, and servers within a group are homogeneous and have exponentially distributed service times that depend on the job type. We seek to minimize the jobs' *Expected queue Waiting time* (EW).

This routing problem arises in different applications, including allocating vehicles to requesters in *Transportation On Demand* (TOD) systems, routing calls to agent groups at call centers, and allocating user tasks to distributed processors (also known as *load balancing*).

In TOD systems such as emergency systems (i.e., ambulances, fire trucks, police patrols, etc.), courier services and taxi networks, the system randomly receives service requests. For each request, a server (i.e., an ambulance, fire truck, police patrol, courier or taxi) is dispatched to the requester's location, serves the requester, finishes the service, returns to its base, becomes available and waits for the next request. In this setting, both requesters and vehicles are geographically distributed. This in turn gives rise to job-and-server dependent service rates where the service rate depends on the distance between the vehicle base and the requester (c.f., Cho *et al.* 2014). For an excellent survey on TOD, please refer to Cordeau *et al.* (2007).

In a call center, calls of different types arrive randomly and a specialized switch called an Automatic Call Distributor (ACD) routes calls to agents. Agents within different groups have different skill sets and consequently different service rates for serving a job, which makes the router's decision challenging. For an excellent survey on *skill-based routing* for call centers, see Gans *et al.* (2003).

In computer systems, a dispatcher distributes jobs generated by users over a set of processors; for an excellent review of *load balancing* see Combé & Boxma (1994). In this context, job-and-server dependent service times can occur for two different reasons, (i) the processors are geographically distributed and the service time is the sum of transferring the job to a processor and the processing time spent on the processor, and (ii) when the processors are of different speeds and jobs are of different types.

In all aforementioned applications, the service time depends both on the jobs and the server. This, in turn, makes the routing problem challenging (Dai & Tezcan, 2008). Thus, both in the literature and in practice, routers commonly apply the FSF policy for this problem (c.f., Mehrotra *et al.*, 2012). We explain FSF and our proposed routing policies in Section 2.4.

Our main contributions are as follows. Combé & Boxma (1994) used a math program to design dynamic policies for a single class, multi-server problem. We take a similar approach for multi-class multi-server systems, and formulate a math program to first compute an optimal static policy. Then, we use the solution to our math program to build policies that take into account the state of the system when routing jobs; we call this step "dynamizing" the static policy. We prove that our math program is convex and thus can be solved efficiently. Our simulation experiments show that our proposed dynamic policy, namely FSFOptXOverflowBlock, beats the well-known FSF policy. While the focus of this study is on designing a dynamic policy, our proposed math program can be used for planning problems such as determining the optimal number of servers for guaranteeing a desired expected waiting time or determining the optimal location of server groups. In Section 2.7, we extend our results

and show that similar dynamic policies can be easily devised for *Expected Sojourn Time* (ES), i.e., the expected time in the system including queue waiting time and service time and the *Expected Throughput* (ET), i.e., the expected number of jobs that are not blocked upon arrival.

The rest of this paper is organized as follows. In Section 2.2, we review the related literature. In Section 2.3, we characterize the optimal static policy. In Section 2.4 we introduce several dynamized routing policies, and in Section 2.5, we conduct computational experiments to compare policies. Finally, we showcase the application of policies for routing fire engines to incident locations in Irvine, California in Section 2.6. Extensions of our model and conclusions follow in Sections 2.7 and 2.8, respectively.

2.2 Literature Review

While most of the research on dynamic policies concerns a single job type, the optimal dynamic policy for routing heterogeneous jobs to heterogeneous servers when the service time depends on the job type and the server group is not known (Mehrotra *et al.*, 2012). In fact, the time to serve a job depends on the routing decision, and we encounter endogenous service times. In the following we review some of the most related papers.

For the special case with one single job type and multiple heterogeneous servers (i.e., single-class multi-server setting), Armony (2005) has shown that the FSF dynamic policy asymptotically minimizes the steady-state expected waiting time in the Halfin-Whitt many-server heavy-traffic regime also known as the Quality and Efficiency Driven regime. Upon a job arrival, FSF routes the job to the fastest idle server, i.e., among server groups that has at least one idle server, the FSF routes to the group with the minimum mean service time for that particular job type. There is a queue for each job type. Upon a service completion,

among all non-empty job queues, the server picks a job from the queue with the minimum mean service time. We discuss all routing policies in Section 2.4. The optimality of FSF has not been shown for a multi-class multi-server system with heterogeneous job types and heterogeneous server groups. Further, our experiments show that FSF is not optimal for the multi-class multi-server setting with heterogeneous job types and heterogeneous server groups. In a seminal work, Mandelbaum & Stolyar (2004) showed that the generalized $c\mu$-rule ($Gc\mu$-rule) asymptotically minimizes convex holding costs for the multi-class multi-server settings. Roughly speaking, the $Gc\mu$-rule is a routing policy that myopically tries to maximize the rate of decrease of the immediate holding cost. However, this policy cannot be applied to linear holding costs, such as expected waiting time. In fact, Dai & Tezcan (2008) discuss that applying the $Gc\mu$-rule to linear holding costs can lead to system load explosion.

In a single-class multi-server setting, Armony & Ward (2010) seek to minimize expected waiting time while considering fairness among servers. They prove that a threshold policy based on the total number of customers in the system is optimal in the Halfin-Whitt many-server heavy-traffic limit regime. We will explain that our method is also capable of handling fairness constraints such as limits on workloads or utilizations.

Under a many-server asymptotic regime, Tezcan & Dai (2010) have shown that the FSF policy is asymptotically optimal for a system with two job types and two server groups, where one group can serve only one job type and the other can serve both (also known as the N-model). But, in their paper, service times are independent of job types (i.e., not job-and-server dependent).

In the absence of an optimal routing policy for multi-class multi-server systems, FSF has been used in the literature for minimizing expected waiting time (c.f., Mehrotra *et al.*, 2012; Chan *et al.*, 2014; Gopalakrishnan *et al.*, 2016). Dai & Tezcan (2008) provide some insight into the challenge of finding the optimal dynamic policy for this problem. We show that our proposed dynamic policies empirically outperform FSF on an extensive set of simulated

instances.

2.3 Optimal Static Policy

In this section we characterize the optimal static policy that minimizes the expected waiting time of an arbitrary job. First, in Section 2.3.1 we describe our multi-class multi-server system. Then, in Section 2.3.2 we formulate a math program for determining the optimal static routing policy. Finally, in Section 2.3.3 we introduce OptXRand, a static routing policy constructed from the solution of our math program.

2.3.1 Framework: Multi-Class Multi-Server Queueing System

In our multi-class multi-server queueing system, jobs of types $i \in I$ arrive according to independent Poisson processes with rates d_i. When a job arrives, the router can either accept or block it. If the job is accepted, it would stay in the system until it is served, i.e., there is no abandonment or retrial following the acceptance stage. Each accepted job must be routed to a server group $j \in FJ_i$, where FJ_i is the set of server groups that are eligible to serve jobs of type i. For completeness, we will denote J as the full set of server groups (henceforth known simply as groups) and FI_j as the set of job types that can be served by group j. In the call center literature, the terms job type and server group are referred to as call type and agent group, respectively. At each server group j, there are k_j identical servers. Service times are independent, each exponentially distributed with mean service time τ_{ij}.

In this setting, we seek to minimize the expected queue waiting time of an arbitrary job, i.e., a random job of any type. A routing policy needs to decide, for each job that arrives and has a certain type i, which server group j that job should be processed on, given the current availability of the servers, and taking into account that the service time τ_{ij} depends on both

the job type i and the server group j.

2.3.2 A Math Program for Determining the Optimal Static Routing Policy

In our math program the decision variable x_{ij} determines the number of jobs of type i to be served by group j per unit time. A job routed to group j which finds all servers in that group busy waits in a queue in front of that server group. For each queue j we assume that (i) the service discipline at each group is First-Come, First-Served (FCFS); and (ii) the system backlogs jobs at the groups until there are available servers to serve the jobs, i.e., no abandonment or retrial exists once a job enters a group's queue. Although jobs are not blocked at the groups (i.e., after being routed to a group), jobs might be blocked upon arrival to job nodes (i.e., before being routed to a group). This allows the routing policy to control the workloads entering the queues. With job-server dependent service times, we notice that the workload of each group is itself a function of the routing policy, i.e., a function of x_{ij}'s.

To construct a math program, we additionally assume static routing so that the model preserves Poisson arrivals at each group. Thus, queues at groups are M/M/k and the delay probability at group j, i.e., the probability that a randomly-chosen job of type i find all servers in group j busy serving other jobs, is computed using the Erlang-C function (Cooper, 1981) defined as,

$$EC(k_j, r_j) = \frac{r_j^{k_j}}{(k_j - 1)(k_j - r_j)} \times \Big[\sum_{n=0}^{k_j - 1} \frac{r_j^n}{n!} + \frac{r_j^{k_j}}{(k_j - 1)!(k_j - r_j)} \Big]^{-1}, \tag{2.1}$$

where symbol (!) is the factorial function, r_j denotes the workload of group j as defined in (2.3), and k_j is the number servers in group j regardless of their status, i.e., busy or available. We expect, however, that our model would also be useful in the context where

service times are not exponential since others (c.f., Kimura 2010) have shown the Erlang-C function tends to be a good approximation for the delay of a M/G/k queue, which has no known closed-form formula.

The arrival rate to group j is the sum of the rates at which we process jobs of the types that this group serves,

$$\lambda_j = \sum_{i \in FI_j} x_{ij} \quad \forall j \in J \qquad . \tag{2.2}$$

The workload of group j is,

$$r_j = \sum_{i \in FI_j} \tau_{ij} x_{ij} \quad \forall j \in J \qquad . \tag{2.3}$$

The mean service time at group j is defined as,

$$\tau_j := \frac{r_j}{\lambda_j} \quad \forall j \in J \qquad . \tag{2.4}$$

The model is schematically depicted in Figure 2.1 with the notation summarized in Table 2.1.

The *Expected Waiting time* (EW), also known as the *Average Speed of Answer* (ASA) for call centers, is the expected time a job spends in the queue before being served. For M/M/k queue at group j, EW is (Hokstad, 1978),

$$EW(\lambda_j, r_j) = EC(k_j, r_j) \frac{r_j}{\lambda_j (k_j - r_j)}, \tag{2.5}$$

where the Erlang-C function $EC(k_j, r_j)$ is defined in (2.1). While the above function measures the expected waiting time of jobs at a single group (or equally a single queue before

Table 2.1: Model notation

Indices and Sets	
$i \in I$	Index for job types
$j \in J$	Index for server groups
F	Set of feasible job type-server group assignments ($i \to j$)
FJ_i	The subset of groups that can serve jobs of type i
FI_j	The subset of job types that group j can serve
Parameters	
d_i	Expected arrival (demand) rate of job type i
τ_{ij}	Mean service time for a server at group j to serve a job of type i
k_j	Number of servers at group j
CF	A number between zero and one indicating the minimum fraction of jobs that should be covered (routed)
Variables	
x_{ij}	Number of jobs of type i to be served by group j per unit time
λ_j	Total number of jobs to be served by group j per unit time
r_j	Workload assigned to group j
τ_j	Mean service time at group j

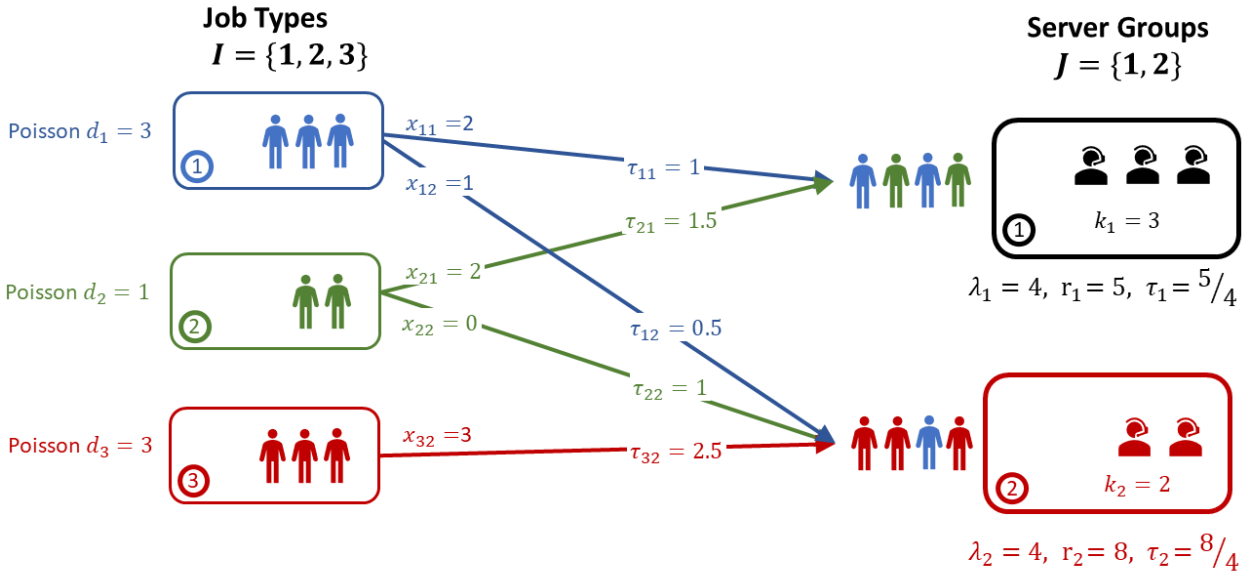


Figure 2.1: Schematic representation of the multi-class multi-server routing system modeled by our math program.

each group), we are interested in an aggregate measure that can help the decision-maker evaluate the performance of the whole system. For this purpose, we define the expected queue waiting time of a random job (or equivalently, an arriving job of any type) denoted by EW_{avg} as,

$$EW_{avg} := \frac{\sum_{j \in J} \lambda_j EW(\lambda_j, r_j)}{\sum_{j \in J} \lambda_j}, \tag{2.6}$$

where the numerator is called the total expected waiting time, EW_{tot},

$$EW_{tot} := \sum_{j \in J} \lambda_j EW(\lambda_j, r_j). \tag{2.7}$$

Consider minimizing EW_{avg} in the math problem $(P1)$ below, which determines x_{ij}'s, the

number of jobs allocated to each group per unit time,

$$
\begin{aligned}
(P1) \quad \min \quad & EW_{avg} \\
\text{s.t.} \quad & \text{Constraints (2.2) and (2.3)}, \\
& r_j \leq k_j \quad \forall j \in J, && (2.8) \\
& \sum_{j \in FJ_i} x_{ij} \leq d_i \quad \forall i \in I, && (2.9) \\
& \text{Optional convex constraints}, && (2.10) \\
& x_{ij} \geq 0 \quad \forall (i,j) \in F. && (2.11)
\end{aligned}
$$

Constraint (2.8) assures the workload at each group does not exceed the number of servers at that group, and is required for the system to be in steady state. If workload exceeds the number of servers the queue would explode. Constraint (2.9) makes sure that for each job type i, the covered jobs are less than or equal to the arrival rate (also called demand) at node i. Because Constraint (2.9) is an inequality, it allows for the possibility that some jobs are blocked upon arrival, i.e., $(P1)$ determines allocation and coverage rate of each job type simultaneously. To preserve Poisson arrivals at the group queues, this blocking is done randomly proportional to x_{ij}'s. Constraint (2.10) indicates that one has the option to impose convex constraints on x_{ij}'s, r_j's and λ_j's. For example, we can make sure that the workload is fairly distributed between groups by imposing lower and upper bounds on r_j's. Or, we can easily embed $(P1)$ into a much larger math program to link additional decisions. Constraint (2.11) makes sure that each x_{ij} is nonnegative.

If we were to minimize EW_{avg} in $(P1)$, we would find that because we did not impose a minimum coverage constraint, the optimal solution would not serve any job and have no congestion. Thus, we add Constraint (2.12), which exogenously defines a global coverage

level,

$$\sum_{j \in J} \lambda_j \geq CF \sum_{i \in I} d_i, \tag{2.12}$$

where $0 \leq CF \leq 1$ is the global coverage factor. A coverage factor of 1, i.e., $CF = 1$, indicates a full coverage of all arriving jobs.

Given a feasible instance of $(P1)$ with Constraint (2.12) in place, then Constraint (2.12) binds at optimality (see Proposition 2.3.1). Indeed, one cannot improve EW_{avg} by increasing the coverage, $\sum_{j \in J} \lambda_j$, beyond the minimum required coverage, $CF \sum_{i \in I} d_i$.

Proposition 2.3.1. *Assuming that problem* $(P1)$ *with Constraint (2.12) is feasible, the coverage Constraint (2.12) is binding.*

Proof. See Appendix I. □

Proposition 2.3.1 indicates that at optimality EW_{avg} becomes,

$$EW_{avg} = \frac{\sum\limits_{j \in J} \lambda_j EW(\lambda_j, r_j)}{\sum_{j \in J} \lambda_j} = \frac{\sum\limits_{j \in J} \lambda_j EW(\lambda_j, r_j)}{CF \sum_{i \in I} d_i} = \frac{EW_{tot}}{CF \sum_{i \in I} d_i}. \tag{2.13}$$

Since $CF \sum_{i \in I} d_i$ is a constant, optimizing EW_{avg} in Problem $(P1)$ with Constraint (2.12) is equivalent to optimizing EW_{tot} with Constraint (2.12). Consequently, although we are most interested in solving $(P1)$ with the EW_{avg} objective and Constraint (2.12), we will do this by solving $(P2)$, defined as

$$\begin{aligned} (P2) \quad \min \quad & EW_{tot} \\ \text{s.t.} \quad & \text{Constraints (2.2), (2.3), (2.8), (2.9), (2.10) and (2.11),} \\ & \sum_{j \in J} \lambda_j = CF \sum_{i \in I} d_i. \end{aligned} \tag{2.14}$$

The following Proposition proves that EW_{tot} is non-linear but convex in r_j, which makes it amenable to numerical optimization solvers. More specifically, since the objective function EW_{tot} is a convex function of the decision variables, and all constraints are linear, $(P2)$ with the EW_{tot} objective is a convex math program, which can be solved by a commercial convex solver such as KNITRO.

Proposition 2.3.2. *$EW_{tot} = \sum_{j \in J} \lambda_j EW(\lambda_j, r_j)$ is convex in r_j and independent of λ_j.*

Proof. See Appendix J. □

In a special case of problem $(P2)$ when there is exactly one server at each group (i.e., $k_j = 1 \ \forall j \in J$), the Erlang-C function $EC(k_j = 1, r_j)$ simplifies to workload r_j, and EW_{tot} becomes,

$$EW_{tot} = \sum_{j \in J} \lambda_j r_j \frac{r_j}{\lambda_j (1 - r_j)} = \frac{r_j^2}{1 - r_j}.$$

Although EW_{tot} is still nonlinear and non-quadratic, in this special case, EW_{tot} is solely a function of r_j's.

In the following, we introduce OptXRand, a static routing policy that uses the solution $\{x_{ij}\}$ that we obtain from solving problem $(P2)$ to route jobs to servers in real-time.

2.3.3 OptXRand: The Optimal Static Policy

Given x_{ij}'s from solving problem $(P2)$, the OptXRand policy works as follows. Upon arrival of a job of type i, with probability $\bar{C}_i/d_i$ the job is blocked, where $\bar{C}_i = d_i - \sum_{j \in J} x_{ij}$ is the number of un-covered jobs of type i. Each non-blocked job is randomly routed to group j with probability $x_{ij} / \sum_{j' \in J} x_{ij'}$. If the job finds all servers at group j busy, it waits in a First-Come, First-Served (FCFS) queue in front of group j with no blocking or abandonment.

Note that Mehrotra *et al.* (2012) implemented a similar policy they call OptXRand, where they obtain routing probabilities x_{ij}'s from solving a slightly different mathematical program. While we minimize EW_{tot} in Problem $(P2)$, Mehrotra *et al.* (2012) maximize total call resolution rate. Since the spirit here is the same, we use the name of their policy. Figure 2.2 schematically illustrates the OptXRand policy.

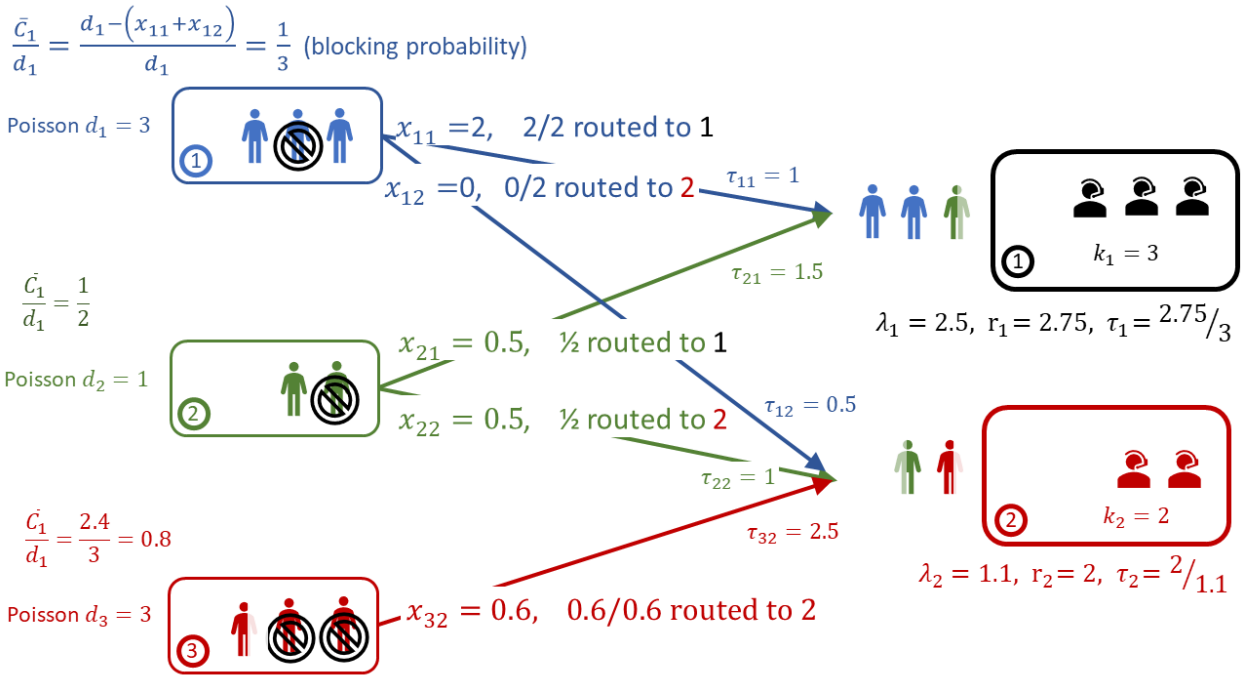


Figure 2.2: Schematic representation of the OptXRand policy.

2.4 Dynamic Routing Policies

We now develop several dynamic routing policies that use the solution from our planning problem $(P2)$ to decide how to assign jobs of each type i to each server group j in real-time. We also describe the *Fastest-Server-First* (FSF) dynamic policy, which is a well-known and common dynamic policy for the multi-class multi-server setting and does not use the solution to $(P2)$ as input (Mehrotra *et al.*, 2012; Armony, 2005). We begin with FSF, since it is both

our benchmark and its logic is incorporated into some of our policies. Finally, in the following section we will empirically compare the performance of all policies.

Recall that to construct the math program ($P2$) in Section 2.3.2, we assumed that queues were at the servers. Putting the queues at the servers allowed us to model each group's queue as an M/M/k queue, which led to expressions that are amenable to convex programming. However, our dynamic policies as well as the FSF policy flips things around, and instead places the queues on the job type side of the graph (i.e., on the left side of the graph depicted in Figure 2.1). This has the effect of postponing the routing decision until the system's status is realized (i.e., until a server becomes available or a job arrives), which enhances the performance of dynamic policies.

2.4.1 FSF Policy

An intuitive solution to the routing problem is the so-called FSF overflow policy. When a job of type i arrives, a job-to-group priority list for that job type determines the order in which the groups are checked for a free server. The priority list is an ordered list of all groups that can serve job i, i.e., $j \in FJ_i$, sorted from smallest to largest service times τ_{ij}. If there are more than one idle servers at the assigned group, we choose the server with the longest idle time, i.e. Longest Idle Server, First (LISF). The policy is called FSF because the fastest server has the highest priority in the list. There is one queue for each job type $i \in I$. If job i finds all groups in FJ_i busy, it will stay in queue i, where it will be served in First-Come, First-Served (FCFS) order. Similarly, when a server in group j becomes free, a group-to-job list for that server group determines the order by which the server picks the next job to serve. The group-to-job list for group j is a list of all job types that server group j can serve, i.e., $i \in FI_j$, sorted in increasing order by τ_{ij}'s. If all queues in the list are empty, then the server stays free. Figure 2.3 schematically illustrates the FSF policy.

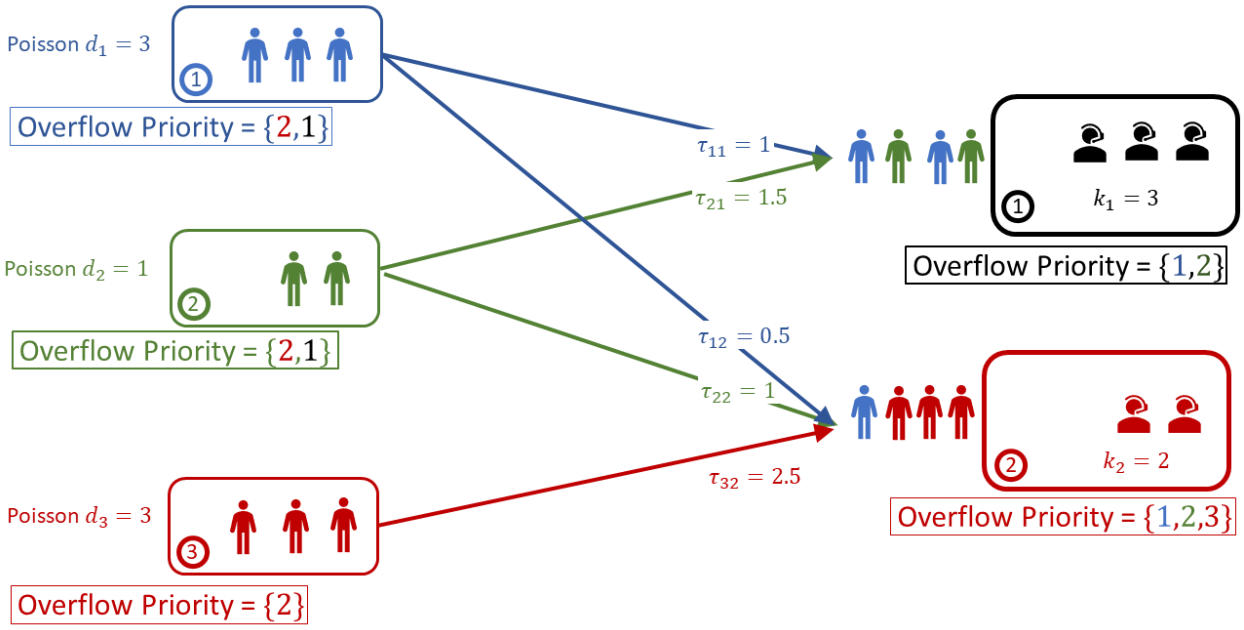


Figure 2.3: Schematic representation of the FSF policy.

2.4.2 FSFBlock Policy

Note that FSF may not perform well because the system accepts all jobs and the system may get overloaded. To address this concern and to have a fair comparison between FSF and other policies that block some jobs, we introduce a policy called FSFBlock that uses the coverage factor CF to randomly block some jobs upon arrival. One should notice that here the blocking probability (i.e., CF) is homogeneous across job types, while in the OptXRand policy the blocking probability is job type-specific (notice the index i in $\bar{C}_i/d_i$ for OptXRand).

2.4.3 OptXOverflow Policy

Like the FSF policy, our OptXOverflow policy is also an overflow policy that uses priority lists to determine routings. However, instead of determining priorities by rank-orders of service time τ_{ij}, OptXOverflow uses the rank-orders of the solution x_{ij} obtained from solving

problem $(P2)$ to sort the list, i.e., job-to-group and group-to-job lists are sorted from largest to smallest x_{ij}. Figure 2.4 schematically illustrates the OptXOverflow policy.

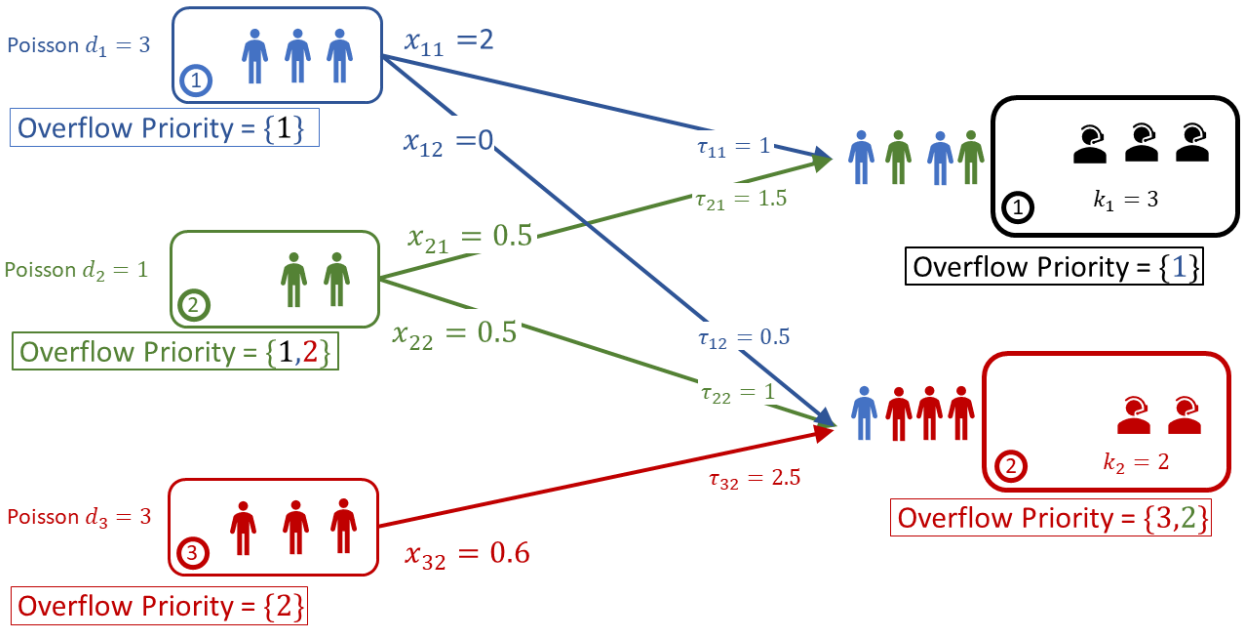


Figure 2.4: Schematic representation of the OptXOverflow policy.

2.4.4 OptXOverflowBlock Policy

This policy is a combination of OptXRand and OptXOverflow. Similar to OptXRand, upon arrival of a job of type i, with probability $\bar{C}_i/d_i$ the job is blocked. Non-blocked jobs are routed according to the priority lists defined by OptXOverflow. Figure 2.5 schematically illustrates the OptXOverflowBlock policy.

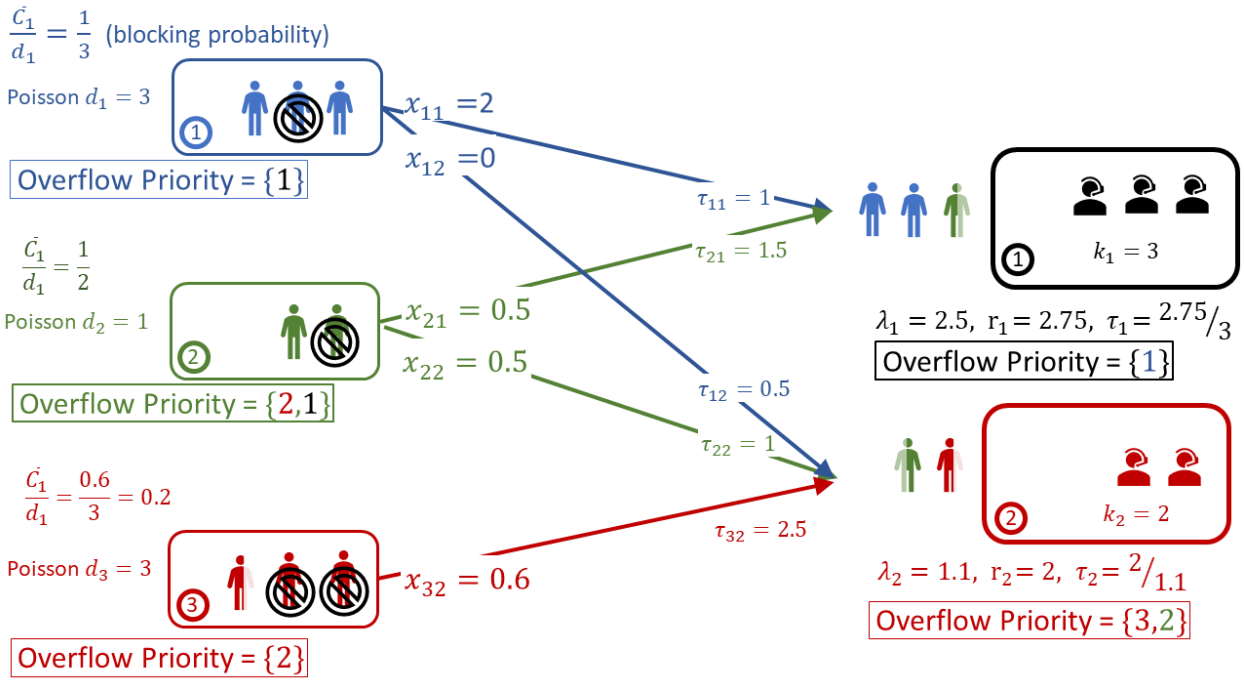


Figure 2.5: Schematic representation of the OptXOverflowBlock policy.

2.4.5 FSFOptXOverflow Policy

This policy is a combination of the FSF and OptXOverflow policies. First, we initialize the priority lists so that they are the same as in the OptXOverflow policy, i.e., groups and jobs are rank-ordered from highest to lowest x_{ij}. Then, we append additional groups (and jobs) to the end of job-to-group (and group-to-job) lists in order of smallest to largest mean service time τ_{ij}, which is consistent with FSF. Thus, priority lists for the FSFOptXOverflow policy are at least as long as lists in the OptXOverflow policy and the FSF policy.Figure 2.6 schematically illustrates the FSFOptXOverflow policy.

2.4.6 FSFOptXOverflowBlock Policy

FSFOptXOverflowBlock is a variant of the FSFOptXOverflow policy in which arrivals of type i are blocked with probability $\bar{C}_i/d_i$. Non-blocked jobs are routed according to the

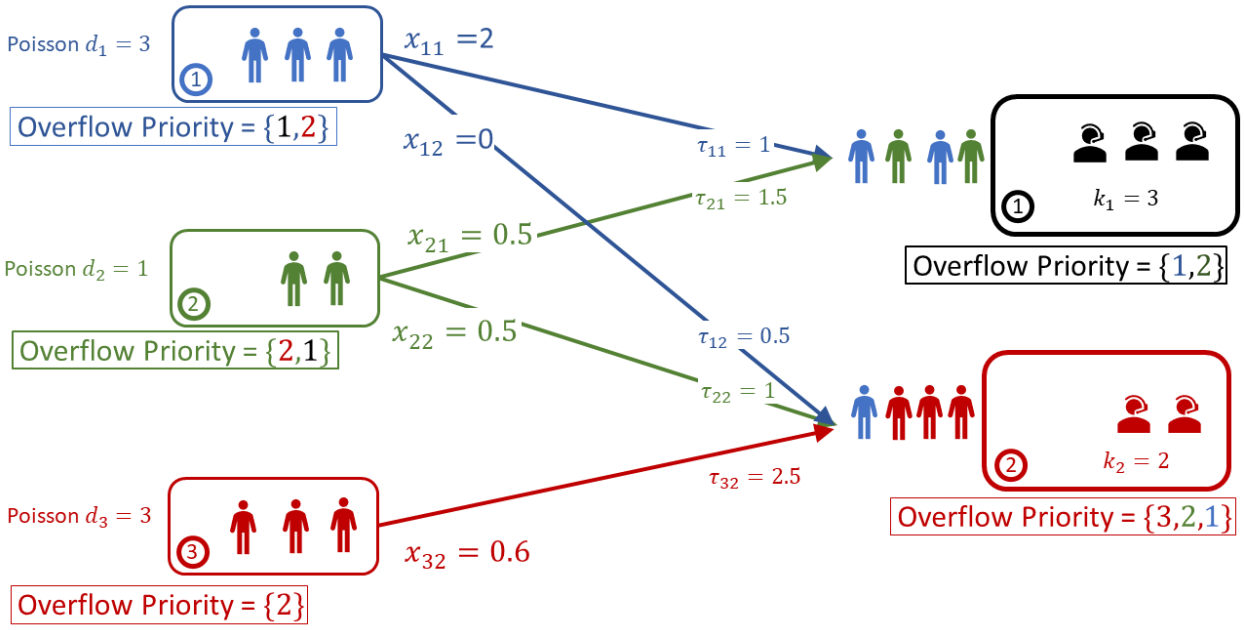


Figure 2.6: Schematic representation of the FSFOptXOverflow policy.

priority lists defined by FSFOptXOverflow. Figure 2.7 schematically illustrates the FSFOptXOverflowBlock policy.

2.5 Experiments

We coded problem ($P2$) in AMPL 2017.01.26 and used a Windows PC with 16 GB of RAM and a quad core CPU with 3.1 GHz. We solve the convex problem ($P2$) using KNITRO 10.1, a commercial solver for convex programs. In Section 2.5.1 we explain how we generate our test instances. Section 2.5.2 discusses the simulation platform and simulation details. We discuss simulation results in Section 2.5.3.

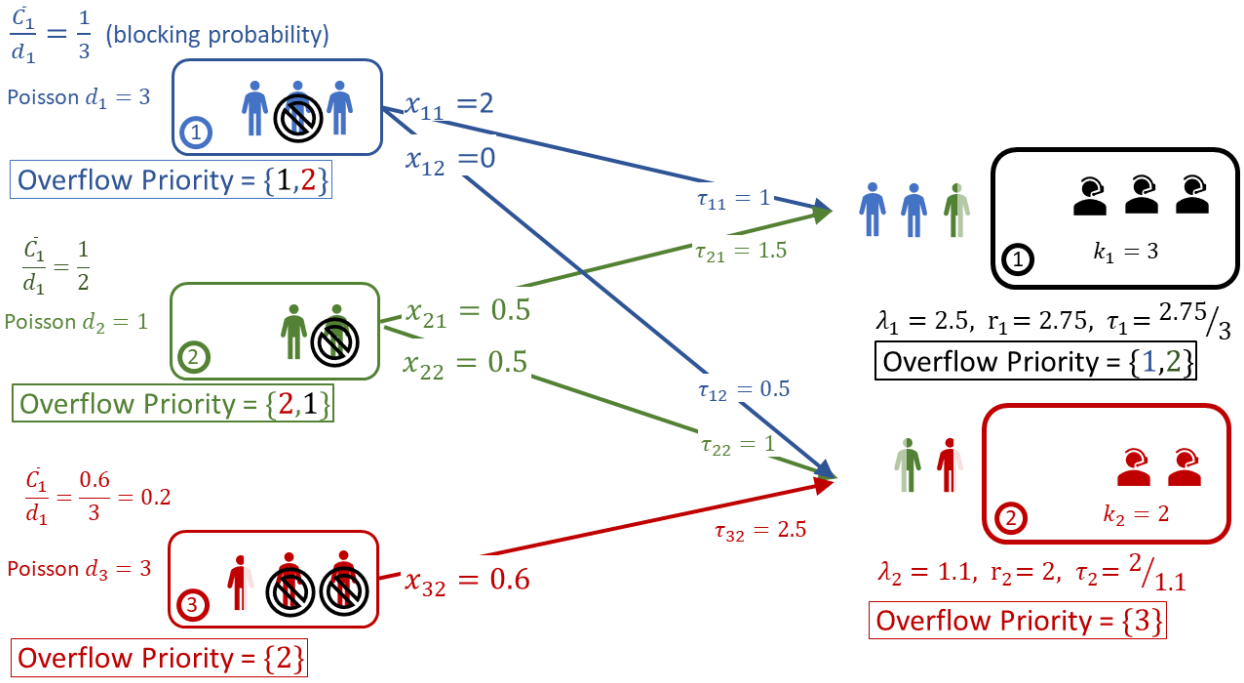


Figure 2.7: Schematic representation of the FSFOptXOverflowBlock policy.

2.5.1 Data

We generate a total of 20 instances in two sets, (i) a *planar* dataset of 6 instances for transportation on demand systems, and (ii) a *non-planar* dataset of 14 instances for call centers and load balancing systems.

For our non-planar instances, we generate τ_{ij}'s randomly from a uniform distribution with support on the interval $[0.5, 3.5]$. For each (i, j) combination, we generate a uniform random number between zero and one; if it's greater than a threshold we keep the link, otherwise we remove it. After removing links we form the adjacency sets FJ_i and FI_j.

For our planar instances, we generate random points for job nodes and group nodes on a 100×100 unit plane. We define a coverage radius of 10 units for each group to determine if a group can serve a job node. If job i is covered by group j, i.e., $i \in FI_j$, the Euclidean distance between i and j gives us the service time, τ_{ij}.

For both datasets, we generate the expected arrival (demand) rate d_i of each job type i and the number of servers k_j at each group j randomly according to the uniform distribution between 1 and 10 and the discrete uniform distribution between 1 and 10, respectively. For many of our instances we consider several coverage factors. In summary, we solve problem ($P2$) and simulate our policies for 42 test cases (i.e., 42 {instance, coverage factor} combinations).

2.5.2 Simulation Platform

We measure the performance of our routing policies using simulation. Specifically, we employ the *ContactCenters* Java library developed by Buist & L'Ecuyer (2005). In our simulation, jobs arrive independently according to Poisson processes, each with arrival rate d_i. Service rates are independent, each exponentially distributed with mean service time τ_{ij}. We run a 14-month simulation and discard the results from the first warm-up month and the fourteenth wrap-up period. We replicate the simulation 10 times and compute the average performance. As the performance metric, we count the total expected waiting time of all admitted jobs, EW_{tot}, and average expected waiting time of a random admitted job, EW_{avg}.

2.5.3 Results

In Tables L.8, L.9 and L.10 we list the results for 42 test cases. Overall, the results indicate that FSFOptXOverflowBlock and OptXOverflowBlock are the best policies. FSFOptXOverflowBlock is the best-performing policy in 22 test cases compared to 19 test cases where OptXOverflowBlock is the best policy. OptXRand is the best policy in 1 test case. In 2 test cases where OptXOverflowBlock is the best policy, the coverage factor is 1, i.e., no blocking occurs so that OptXOverflowBlock is the same as OptXOverflow.

In all 42 test cases, the FSF and FSFBlock policies are dominated by other policies, particularly OptXRand, OptXOverflowBlock and FSFOptXOverflowBlock. This is because FSF and FSFBlock are myopic and decentralized, i.e., each job chooses the fastest available server without considering other jobs, while policies that use the solution to problem ($P2$) consider the overall system congestion and may route a job to a server that is not the fastest available server, keeping the fastest available server for a future incoming job or even a job of a different type. Indeed, these policies try to balance server workloads to benefit the whole system, and are less greedy.

We also notice that blocking jobs upon arrival is crucial, as results suggest that policies that limit the workload of servers, i.e., FSFBlock, OptXOverflowBlock, and FSFOptXOverflowBlock perform better than similar policies without blocking, i.e., FSF, OptXOverflow, and FSFOptXOverflow, respectively. One should note that finding the optimal blocking level is not trivial because service rates are job-server dependent and servers' workloads depend on the routing. Recall that FSFBlock blocks jobs using the homogeneous coverage factor (i.e., blocking probability) CF. However, OptXRand, OptXOverflowBlock, and, FSFOptXOverflowBlock use the solution to problem ($P2$) to block jobs, which allows the blocking probability, $\bar{C}_i/d_i$, to be job type-specific. As our results show, job type-specific blocking rates are important for producing quality solutions.

Our simulation results show that the solution to our math program ($P2$) is useful for building "dynamized" routing policies for online settings. Particularly, FSFOptXOverflowBlock and OptXOverflowBlock are both well-performing policies. This suggests that there is a strong benefit from hybridizing a static policy computed by solving a math program, which effectively apportions aggregate workloads from heterogeneous jobs types to heterogeneous servers, with a dynamic policy that is provably near-optimal for special cases (i.e., only one job type) and takes into account real-time state information about the availability of servers.

2.6 Fire Stations Case

To illustrate the applicability of our methodology, we now assign incidents to fire stations in Irvine, CA. We describe the data in Section 2.6.1, introduce benchmark routing policies in Section 2.4, illustrate key routing policies in Section 2.6.2, and finally compare routing policies in Section 2.6.3.

2.6.1 Data

For emergency calls, the Orange County Fire Authority (OCFA) dispatches vehicles to incident locations. We applied our model to the 11 OCFA stations in Irvine, a 66-square-mile city in Orange county with a population of 212,375 (US Department of Commerce: United States Census Bureau, 2010a) and 13 dedicated fire trucks (Orange County Fire Authorities, 2014). In 2010, OCFA received 85,212 emergency calls; we allocate these to 1,822 census block groups in Orange County (OC) proportional to their population according to the 2010 US census (US Department of Commerce: United States Census Bureau, 2010b). We assume that the incidents occur at the centroid of census block groups depicted in Figure 2.8. Among the 1,822 block groups in OC, 117 have centroids that fall in Irvine (see blue and orange block groups in Figure 2.8), and 7 have centroids that do not fall in Irvine (see red block groups in Figure 2.8). Since incident rates are low relative to station capacities, the performance of different routing policies are similar. To compare routing policies, we create congestion in the system by increasing incident rates by a factor of 20. It should be noted that in this paper our goal is to compare the static OptXRand policy with the dynamic policies we developed in Section 2.4 using non-synthetic data, not to provide policy recommendations for OCFA. Indeed, several factors, including the need to cover commercial and industrial facilities with diverse risk portfolios, as well as the need to protect against large-scale wildfires which may encroach on entire subdivisions, may indeed require significant excess capacity compared to

our rough estimates.

The service time τ_{ij} is the time it takes for a vehicle to travel from its station j to the incident location (i.e., the centroid of the block group i), serve the incident, return to its station j and become ready for the following incident. The travel time is calculated based on the street maps in Irvine using the mapping software ArcGIS 10. In computing the driving time, we consider the speed limit of the streets, the traffic direction and other traffic flow constraints. OCFA standards indicate that a fire engine should arrive at the incident's location within an acceptable waiting time of 8 minutes 45 seconds (Orange County Fire Authorities, 2014). Using this acceptable waiting time, we build sets FJ_i and FI_j, i.e., block group i is covered by station j if centroid i is within 8 minutes 45 seconds driving distance of j. There are two centroids that are not within the 8 minutes 45 seconds driving distance of any fire station and thus are not considered in our experiment (see light red block groups in Figure 2.8).

2.6.2 Illustrations of Key Routing Policies

We illustrate job-to-group assignments for the FSF policy in Figure 2.9. To show the map clearly, we only draw the first three assignment priorities in the priority lists, even though there are up to seven. Note that the nearest station is not necessarily the station with the minimum Euclidean distance. The lines in our maps show the assignment of the block groups to the stations, not the minimum distance.

We depict the OptXRand policy in Figure 2.10. Recall that for the OptXRand policy, when an incident in block group i occurs it is randomly routed to $j \in FJ_i$ according to the x_{ij}'s. In practice, such random assignments may be undesirable. However, we can build an equivalent map that is not subject to randomization but corresponds to the OptXRand policy by subdividing block groups into areas proportional to the x_{ij}'s. This is explained in detail in Appendix K.

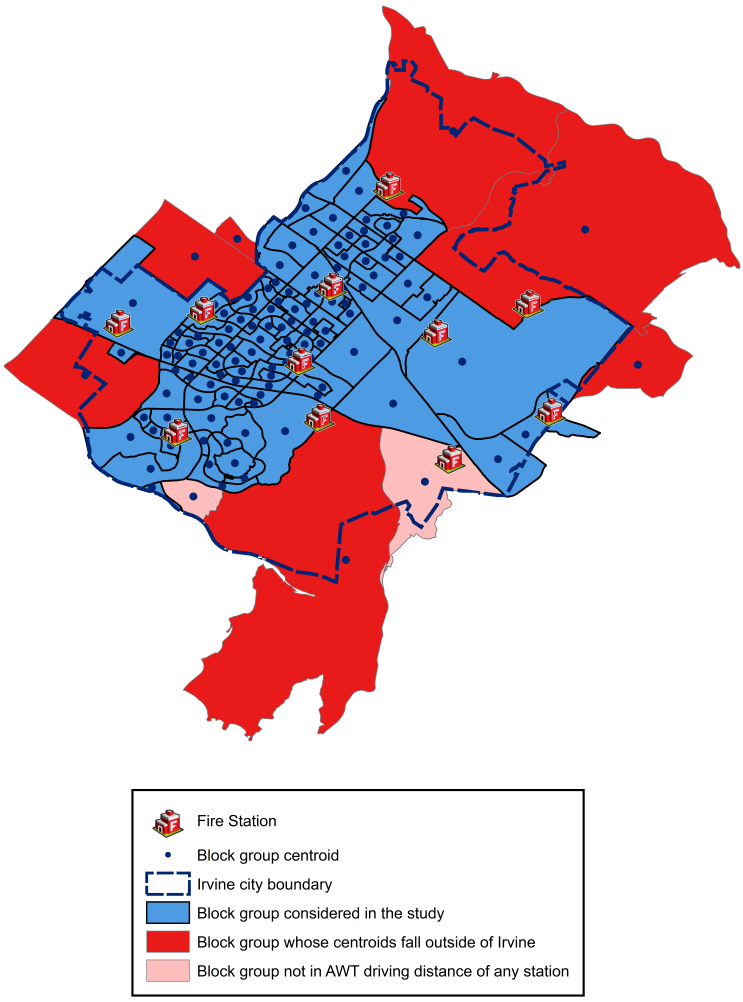


Figure 2.8: Irvine fire stations and the census block groups considered in the study

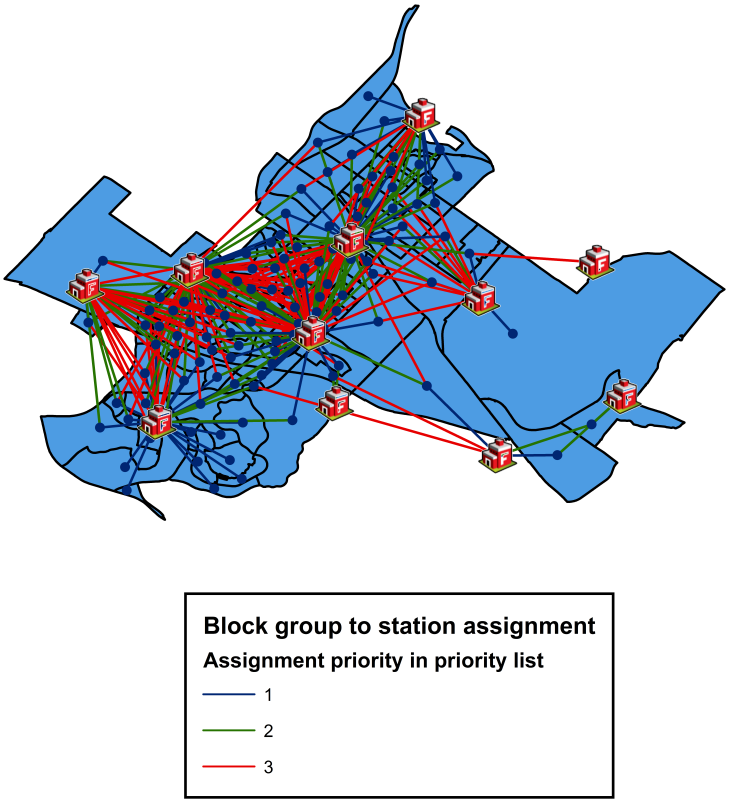


Figure 2.9: Irvine Block groups assigned to stations based on the FSF policy

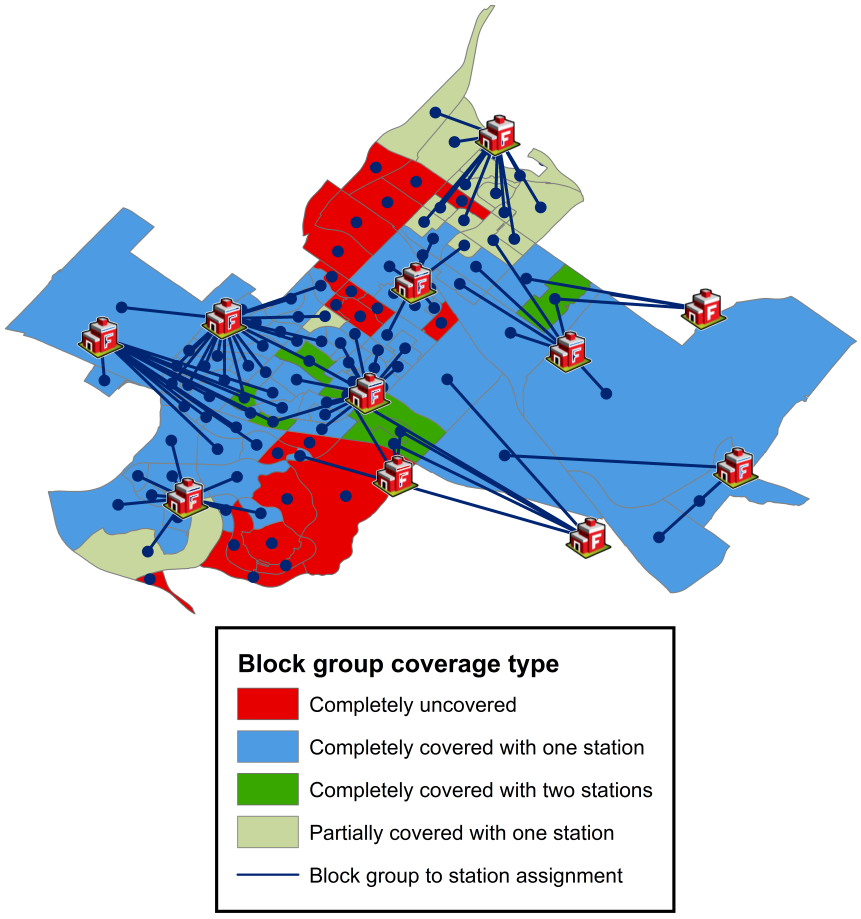


Figure 2.10: Block groups assigned to stations based on the OptXRand policy

Finally we depict FSFOptXOverflowBlock in Figure 2.11, which shows the first three priorities for each block group. Other policies can be mapped similarly.

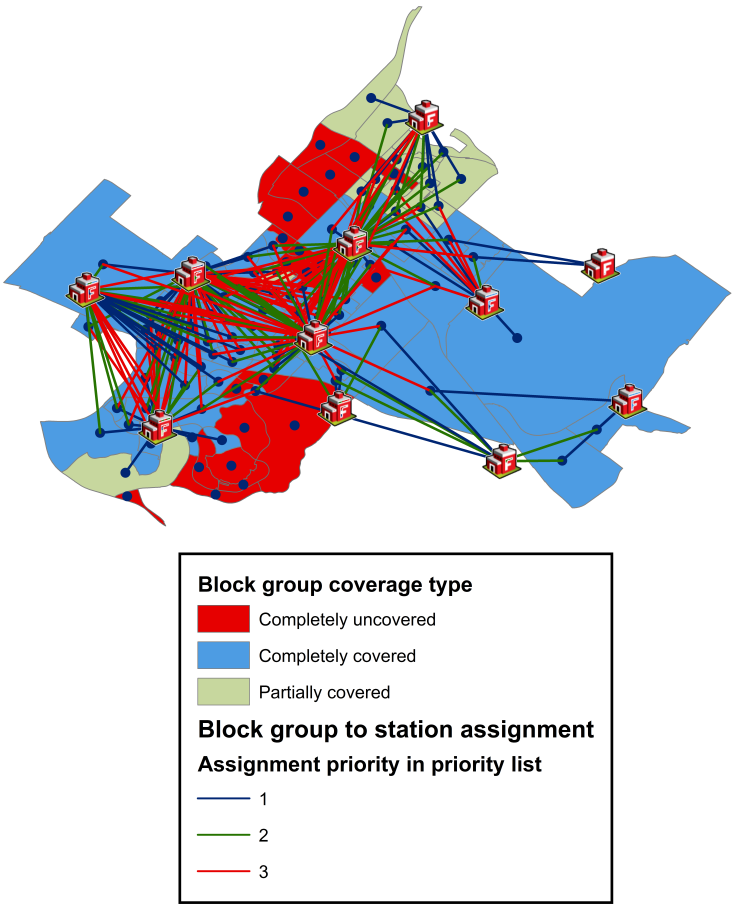


Figure 2.11: Block groups assigned to stations based on the FSFOptXOverflowBlock policy

2.6.3 Results

We simulate each of the dynamic policies mentioned in Section 2.4 as well as OptXRand. The results listed in Table 2.2 show that the FSFOptXOverflowBlock policy is the best routing policy, followed closely by OptXOverflowBlock. This result is consistent with our simulation results in Section 2.5. Both FSFOptXOverflowBlock and OptXOverflowBlock significantly outperform FSF and FSFBlock, which do not use the solution to problem ($P2$). Also, FSFBlock performs slightly better than FSF because it blocks some jobs and thus

controls the system's load. Comparing OptXOverflowBlock and FSFBlock suggests that we benefit from blocking according to x_{ij}'s from solving problem ($P2$) instead of the global coverage factor, CF. We notice that OptXRand is dominated by all other policies except for the FSF policy. This affirms the importance of constructing policies that use the state of the system to route jobs to servers.

Table 2.2: Simulation results for Irvine fire stations case

Routing Policy	Performance Measure	Coverage Factor		
		0.8	0.85	0.9
FSF	EW_{tot}		6.33[1]	
	EW_{avg}		0.51	
FSFBlock	EW_{tot}	6.14	6.24	6.31
	EW_{avg}	0.49	0.50	0.50
OptXRand	EW_{tot}	5.40	6.23	8.20
	EW_{avg}	0.45	0.51	0.74
OptXOverflow	EW_{tot}	5.37	5.93	7.87
	EW_{avg}	0.44	0.49	0.75
FSFOptXOverflow	EW_{tot}	5.35	5.68	6.16
	EW_{avg}	0.43	0.46	0.56
OptXOverflowBlock	EW_{tot}	0.82	1.28	1.85
	EW_{avg}	0.08	0.13	0.17
FSFOptXOverflowBlock	EW_{tot}	**0.82**[2]	**1.24**	**1.81**
	EW_{avg}	**0.08**	**0.12**	**0.16**

[1] FSF policy does not have a coverage factor (that is, it does not block jobs).
[2] **Bold face**: Best routing policy in the respective coverage factor.

2.7 Extensions

Our methodology, which is based on solving a math program to produce an optimal static policy $\{x_{ij}\}$ and then "dynamizing" this policy using job-to-group and group-to-type overflow lists can also be used in a number of other common practical settings. This involves a suitable modification of the planning problem ($P2$), which we suggest for tractability should remain a convex program. For example, our methodology can be extended to optimize *Expected*

System time (ES), i.e., the sum of queue time and service time, and *Expected Throughput* (ET), i.e., the expected number of jobs that are not blocked upon arrival. The remainder of this section confirms that the mathematical expressions needed to optimize ES and ET are indeed convex.

The aggregate measure ES_{tot} is defined as follows,

$$\begin{aligned} ES_{tot} &:= \sum_{j \in J} \lambda_j ES(\lambda_j, r_j) \\ &= \sum_{j \in J} \lambda_j EW(\lambda_j, r_j) + \sum_{(i,j) \in F} \tau_{ij} x_{ij} \\ &= EW_{tot} + \sum_{j \in J} r_j. \end{aligned}$$

Notice that since EW_{tot} is convex in $\{r_j\}$ (see Proposition 2.3.2), adding the linear term $\sum_{j \in J} r_j$ preserves the convexity for ES_{tot}.

Regarding ET, we keep all the assumptions stated in Section 2.3 except the backlogging assumption. Indeed, jobs that find all k servers busy are blocked with no retrial and the queues are M/M/k/k. This allows us to use the Erlang-B function to measure the blocking probability,

$$EB(k, r) = \frac{r^k/k!}{\sum_{n=0}^{k} r^n/n!}.$$

Expected throughput, the probability that an arbitrary job is not blocked, is defined as,

$$ET(r) := 1 - EB(k, r).$$

The aggregate measure ET_{tot} is then defined as,

$$ET_{tot} := \sum_{j \in J} \lambda_j ET(r_j).$$

Harel (1990) proved that $\lambda EB(k, r)$ is jointly convex in λ and r with fixed k. Thus, ET_{tot} is jointly convex in $\{\lambda_j\}$ and $\{r_j\}$ and we can solve problem $(P2)$ with the ET_{tot} objective function. When optimizing ET_{tot}, there is an endogenous trade-off between covering more jobs (i.e., increasing $\sum_{j \in J} \lambda_j$) and improving the performance measure ET_{tot}, and thus coverage and routing can be simultaneously optimized. This indicates that one can solve problem $(P2)$ with ET_{tot} without the coverage constraint defined in (2.14).

As well, one may want to consider using planning models which have the performance measure in a constraint rather than in the objective function. For example, in problem $(P3)$ below we seek to maximize the number of jobs covered subject to a performance measure constraint. For example, one might want to maximize the number of covered jobs such that the average expected waiting time does not exceed 10 minutes.

$$\begin{aligned} (P3) \quad \max \quad & \sum_{j \in J} \lambda_j \\ \text{s.t.} \quad & \text{Constraints (2.2), (2.3), (2.8), (2.9), (2.10), (2.11),} \\ & EW_{avg} \leq \alpha, \end{aligned}$$

where α is the acceptable threshold for the expected waiting time. Problem $(P3)$ is a convex math program and can be solved by commercial solvers (e.g., KNITRO).

2.8 Conclusions

Expected waiting time is an important performance measure for transportation networks and call centers. We showed that for multi-class multi-server systems, this performance measure is convex in the arrival rate and workload. While our math programming model is useful for planning purposes and can be embedded in larger optimization problems, we showed that its solution can also be used to construct good routing policies for real-time routing. We used the solution to our planning problem with expected waiting time objective function to build static and dynamic routing policies. Our experiments and fire station case showed that we can produce a well-performing dynamic policy, FSFOptXOverflowBlock, that outperforms the popular Fastest-Server-First policy.

We showed that our method can be easily applied to other queue performance measures, namely, expected system time and expected throughput. Moreover, performance measures can appear in constraints as well as the objective function; for example, a limit for the queue waiting time.

Our study is valuable because it designs dynamic policies that significantly outperform the FSF policy, which is currently used in the literature and in practice. These dynamic policies are overflow policies that are easy to implement in real-time. Future research directions based on our work include (i) designing dynamic routing policies for other performance measures and (ii) embedding our math program into larger planning problems such as staffing and scheduling problems.

Bibliography

Abate, Joseph, Choudhury, Gagan L, & Whitt, Ward. 1995. Exponential approximations for tail probabilities in queues, I: waiting times. *Operations Research*, **43**(5), 885–901.

Armony, Mor. 2005. Dynamic routing in large-scale service systems with heterogeneous servers. *Queueing Systems*, **51**(3-4), 287–329.

Armony, Mor, & Ward, Amy R. 2010. Fair dynamic routing in large-scale heterogeneous-server systems. *Operations Research*, **58**(3), 624–637.

Atlason, Júlíus, Epelman, Marina A, & Henderson, Shane G. 2008. Optimizing call center staffing using simulation and analytic center cutting-plane methods. *Management Science*, **54**(2), 295–309.

Buist, Eric, & L'Ecuyer, Pierre. 2005. A Java library for simulating contact centers. *Pages 556–565 of: Proceedings of the 37th conference on Winter simulation*. Winter Simulation Conference.

Byrd, Richard H, Nocedal, Jorge, & Waltz, Richard A. 2006. KNITRO: An integrated package for nonlinear optimization. *Pages 35–59 of: Large-scale nonlinear optimization*. Springer.

Cezik, Mehmet Tolga, & L'Ecuyer, Pierre. 2008. Staffing multiskill call centers via linear programming and simulation. *Management Science*, **54**(2), 310–323.

Chan, Wyean, Koole, Ger, & L'Ecuyer, Pierre. 2014. Dynamic call center routing policies using call waiting and agent idle times. *Manufacturing & Service Operations Management*, **16**(4), 544–560.

Cheong, KiJu, Kim, JaeJon, & So, SoonHu. 2008. A study of strategic call center management: relationship between key performance indicators and customer satisfaction. *European Journal of Social Sciences*, **6**(2), 268–276.

Cho, Soo-Haeng, Jang, Hoon, Lee, Taesik, & Turner, John. 2014. Simultaneous location of trauma centers and helicopters for emergency medical service planning. *Operations Research*, **62**(4), 751–771.

Combé, MB, & Boxma, Onno J. 1994. Optimization of static traffic allocation policies. *Theoretical Computer Science*, **125**(1), 17 – 43.

Cooper, Robert B. 1981. *Introduction to Queueing Theory*. North Holland.

Cordeau, Jean-François, Laporte, Gilbert, Potvin, Jean-Yves, & Savelsbergh, Martin WP. 2007. Chapter 7 Transportation on Demand. *Pages 429 – 466 of:* Barnhart, Cynthia, & Laporte, Gilbert (eds), *Transportation*. Handbooks in Operations Research and Management Science, vol. 14. Elsevier.

Dai, J. G., & Tezcan, Tolga. 2008. Optimal control of parallel server systems with many servers in heavy traffic. *Queueing Systems*, **59**(2), 95–134.

Department of Health. 2015. *Urgent and emergency care services in England*. Accessed: 2016-09-30.

Ephremides, Anthony, Varaiya, P, & Walrand, Jean. 1980. A simple dynamic routing problem. *IEEE transactions on Automatic Control*, **25**(4), 690–693.

Fitch, Jay. 2005. Response times: myths, measurement & management. *JEMS: a journal of emergency medical services*, **30**(9), 47–56.

Floudas, Christodoulos A, & Pardalos, Panos M. 2011. *State of the Art in Global Optimization: Computational Methods and Applications*. Nonconvex Optimization and Its Applications. Springer US.

Gans, Noah, & Zhou, Yong-Pin. 2003. A call-routing problem with service-level constraints. *Operations Research*, **51**(2), 255–271.

Gans, Noah, Koole, Ger, & Mandelbaum, Avishai. 2003. Telephone call centers: Tutorial, review, and research prospects. *Manufacturing & Service Operations Management*, **5**(2), 79–141.

Gopalakrishnan, Ragavendran, Doroudi, Sherwin, Ward, Amy R, & Wierman, Adam. 2016. Routing and staffing when servers are strategic. *Operations Research*, **64**(4), 1033–1050.

Grassmann, W. 1983. The convexity of the mean queue size of the M/M/c queue with respect to the traffic intensity. *Journal of Applied Probability*, 916–919.

Harel, Arie. 1990. Convexity properties of the Erlang loss formula. *Operations Research*, **38**(3), 499–505.

Hokstad, Per. 1978. Approximations for the M/G/m queue. *Operations Research*, **26**(3), 510–523.

Hordijk, Arie, & Koole, Ger. 1990. On the Optimality of the Generalized Shortest Queue Policy. *Probability in the Engineering and Informational Sciences*, **4**(10), 477–487.

Kimura, Toshikazu. 2010. The M/G/s Queue. *In:* Cochran, James J., Cox, Louis A., Keskinocak, Pinar, Kharoufeh, Jeffrey P., & Smith, J. Cole (eds), *Wiley Encyclopedia of Operations Research and Management Science*. John Wiley & Sons, Inc.

Koole, Ger. 1999. On the static assignment to parallel servers. *IEEE Transactions on Automatic Control*, **44**(8), 1588–1592.

Lee, Hau Leung, & Cohen, Morris A. 1983. A note on the convexity of performance measures of M/M/c queueing systems. *Journal of Applied Probability*, **20**(4), 920–923.

Liu, Zhen, & Towsley, Don. 1994. Optimality of the Round-Robin Routing Policy. *Journal of Applied Probability*, **31**(2), pp. 466–475.

Louwers, Dirk, Kip, Bert J, Peters, Edo, Souren, Frans, & Flapper, Simme Douwe P. 1999. A facility location allocation model for reusing carpet materials. *Computers & Industrial Engineering*, **36**(4), 855–869.

Mandelbaum, Avishai, & Stolyar, Alexander L. 2004. Scheduling Flexible Servers with Convex Delay Costs: Heavy-Traffic Optimality of the Generalized cÎŒ-Rule. *Operations Research*, **52**(6), 836–855.

Mehrotra, Vijay, Ross, Kevin, Ryder, Geoff, & Zhou, Yong-Pin. 2012. Routing to manage resolution and waiting time in call centers with heterogeneous servers. *Manufacturing & Service Operations Management*, **14**(1), 66–81.

National Fire Protection Association. 2004. Standard for the Organization and Deployment of Fire Suppression, Emergency Medical Administration Operations, and Special Operations to the Public by Career Fire Departments. National Fire Protection Association, Document No. 1710. *Quincy, MA*.

Nourbakhsh, Vahid, & Turner, John. 2018. *Dynamized Routing Policies for Minimizing Expected Waiting Time in a Multi-Class Multi-Server System.* Working Paper Available at SSRN: `https://ssrn.com/abstract=3257582`.

Orange County Fire Authorities. 2014. *Standards of Coverage and Deployment Plan.* Accessed: 2016-09-30.

Pichitlamken, Juta, Deslauriers, Alexandre, L'Ecuyer, Pierre, & Avramidis, Athanassios N. 2003. Modelling and simulation of a telephone call center. *Pages 1805–1812 of: Proceedings of the 35th conference on Winter simulation: driving innovation.* Winter Simulation Conference.

Pot, Auke, Bhulai, Sandjai, & Koole, Ger. 2008. A simple staffing method for multiskill call centers. *Manufacturing & Service Operations Management*, **10**(3), 421–428.

Sahinidis, Nikolaos V. 2014. *BARON 14.3.1: Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual.

Tawarmalani, Mohit, & Sahinidis, Nikolaos V. 2005. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, **103**, 225–249.

Tezcan, Tolga, & Dai, JG. 2010. Dynamic control of N-systems with many servers: Asymptotic optimality of a static priority policy in heavy traffic. *Operations Research*, **58**(1), 94–110.

US Department of Commerce: United States Census Bureau. 2010a. *Quick Facts, Irvine city, California*. Accessed: 2016-09-30.

US Department of Commerce: United States Census Bureau. 2010b. *US Census Block Data*. Accessed: 2016-09-30.

Appendices

A Proof of Proposition 1.3.1: Binding Coverage for SL_{tot}

Proof. Proof by contradiction. Assume that for the optimal solution $\left\{x_{ij}^*\right\}$ with the corresponding arrival rate $\left\{\lambda_j^*\right\}$ and workload $\left\{r_j^*\right\}$ the minimum coverage constraint (1.13) is not binding, i.e.,

$$\sum_{j\in J}\lambda_j^* > C_f \sum_{i\in I} d_i.$$

Construct binding solution $\{\bar{x}_{ij}\}$ with corresponding $\left\{\bar{\lambda}_j\right\}$ and $\{\bar{r_j}\}$ as follows,

$$\bar{x}_{ij} = \beta x_{ij}^* \quad \forall (i,j) \in F,$$

where,

$$\beta = \frac{C_f \sum_{i\in I} d_i}{\sum_{j\in J}\lambda_j^*}.$$

Note that $0 \leq \beta < 1$ and $\sum_{j \in J} \bar{\lambda_j} = C_f \sum_{i \in I} d_i$. Then,

$$\bar{\lambda}_j = \sum_{i \in I} \bar{x}_{ij} = \sum_{i \in I} \beta x^*_{ij} = \beta \lambda^*_j < \lambda^*_j \quad \forall j \in J, \tag{A.1}$$

$$\bar{r}_j = \sum_{i \in I} \tau_{ij} \bar{x}_{ij} = \sum_{i \in I} \tau_{ij} \beta x^*_{ij} = \beta r^*_j < r^*_j \quad \forall j \in J, \tag{A.2}$$

and

$$\bar{\tau}_j = \frac{\bar{r}_j}{\bar{\lambda}_j} = \frac{\beta r^*_j}{\beta \lambda^*_j} = \tau^*_j \quad \forall j \in J.$$

Lemma A.1 proves that $SL(\lambda, r)$ is strictly decreasing in r with fixed τ. Also, with fixed τ and since $r = \tau\lambda$, $SL(\lambda, r)$ is strictly decreasing in λ. Thus,

$$\begin{aligned} SL(\bar{\lambda}_j, \bar{r}_j) &> SL(\lambda^*_j, \bar{r}_j) \\ &> SL(\lambda^*_j, r^*_j) \quad \forall j \in J \end{aligned}$$

Denote SL_{avg} corresponding to solutions $\left\{x^*_{ij}\right\}$ and $\{\bar{x}_{ij}\}$ by SL^*_{avg} and $\overline{SL}_{avg}$, respectively. Using inequalities (A.1) and (A.2), the below inequality follows,

$$\begin{aligned} \overline{SL}_{avg} &= \frac{\sum_{j \in J} \bar{\lambda}_j SL(\bar{\lambda}_j, \bar{r}_j)}{\sum_{j \in J} \bar{\lambda}_j} \\ &= \frac{\sum_{j \in J} \lambda^*_j SL(\bar{\lambda}_j, \bar{r}_j)}{\sum_{j \in J} \lambda^*_j} \\ &> \frac{\sum_{j \in J} \lambda^*_j SL(\lambda^*_j, r^*_j)}{\sum_{j \in J} \lambda^*_j} = SL^*_{avg}, \end{aligned}$$

which contradicts the optimality assumption for solution $\left\{x^*_{ij}\right\}$ with objective value SL^*_{avg}.

□

Lemma A.1. *Service level function $SL(\lambda, r)$ is strictly decreasing in workload r with fixed*

mean service time τ.

Proof. Lee & Cohen (1983) have proved that Erlang-C formula $C(k,r)$ is strictly increasing in workload r. Also, the conditional delay probability, $e^{-(\frac{k-r}{\tau})T}$, is positive and strictly increasing in workload r with positive acceptable waiting time, i.e., $T > 0$, and fixed mean service time τ. Thus, $C(k,r)e^{-(\frac{k-r}{\tau})T}$ as a product of two positive increasing functions is strictly increasing in workload r. Finally, $SL(\lambda,r) = 1-C(k,r)e^{-(\frac{k-r}{\tau})T}$ is strictly decreasing in workload r with fixed mean service time τ. □

B Proof of Proposition 1.3.2: Concavity of SL_{tot}

In an M/M/1 queue, the delay probability $P[W > 0] = C(k,r)$ simplifies to workload r, i.e., $P[W > 0] = C(k,r) = C(1,r) = r$. Also with acceptable waiting time of zero, i.e., $T = 0$, the conditional delay probability $P[W > T|W > 0]$ becomes one. Thus, $SL(\lambda,r) = 1 - P[W > 0]P[W > T|W > 0] = 1 - r$ and the objective SL_{tot} simplifies to,

$$SL_{tot} = \sum_{j\in J} \lambda_j SL(\lambda_j, r_j) \tag{B.3}$$

$$= \sum_{j\in J} \lambda_j - \sum_{j\in J} \lambda_j r_j \tag{B.4}$$

$$= \sum_{j\in J} \lambda_j - \sum_{j\in J} w_j, \tag{B.5}$$

with bilinear terms $w_j = \lambda_j r_j \ \forall j \in J$. Since the first term $\sum_{j\in J} \lambda_j$ is linear in $\{\lambda_j\}$, to investigate the joint convexity of SL_{tot} in $\{\lambda_j\}$ and $\{r_j\}$ it suffices to investigate the joint convexity of each bilinear term w_j. Generally, bilinear terms are non-convex (c.f. Floudas & Pardalos 2011). Lemma B.1 proves that if $\tau_{ij} = \tau_j \ \forall i \in FI_j$ the bilinear term w_j is jointly convex in λ_j and r_j. Lemma B.2 proves that the same condition is necessary for convexity. Regarding the pseudo-concavity of SL_{tot} in $\{\lambda_j\}$ and $\{r_j\}$, again it suffices to investigate the

pseudo-convexity of each bilinear term w_j. Lemma C.1 shows that the bilinear term w_j is not pseudo-convex in λ_j and r_j.

Lemma B.1. *For a given group j, the bilinear term $w_j = \lambda_j r_j$ is jointly convex in $\{\lambda_j\}$ and $\{r_j\}$ if $\tau_{kj} = \tau_j \ \forall k \in FI_j$.*

By performing the substitution for $\lambda_j = \sum_{i \in FI_j} x_{ij}$ and $r_j = \sum_{i \in FI_j} \tau_{ij} x_{ij}$, the bilinear term becomes,

$$w_j = \lambda_j r_j = \sum_{i \in FI_j} x_{ij} \sum_{i \in FI_j} \tau_{ij} x_{ij} \quad \forall j \in J.$$

For each group $j \in J$, since $\tau_{kj} = \tau_j \ \forall k \in FI_j$, the bilinear is,

$$w_j = \sum_{i \in FI_j} x_{ij} \sum_{i \in FI_j} \tau_{ij} x_{ij} = \tau_j \sum_{i \in FI_j} x_{ij} \sum_{i \in FI_j} x_{ij} = \tau_j [\sum_{i \in FI_j} x_{ij}]^2 = \tau_j \lambda_j^2,$$

which is a quadratic convex function. □

Lemma B.2. *For a given group j, if $\exists k, l \in FI_j$ such that $\tau_{kj} \neq \tau_{lj}$ and τ_{kj}, $\tau_{lj} > 0$, the Hessian matrix of the bilinear term $w_j = \lambda_j r_j = \sum_{i \in FI_j} x_{ij} \sum_{i \in FI_j} \tau_{ij} x_{ij}$ is indefinite and the w_j is non-convex in x_{ij}.*

Proof. We suppress index j in w_j and assume that the set FI_j is of length n. Then, the bilinear term w can be expanded as follows,

$$\begin{aligned} w = \sum_{i=1}^{n} x_i \sum_{i=1}^{n} \tau_i x_i = \tau_1 x_1^2 \ &+(\tau_1 + \tau_2) x_1 x_2 + \cdots \ +(\tau_1 + \tau_n) x_1 x_n \\ &+(\tau_2 + \tau_3) x_2 x_3 + \cdots \ +(\tau_2 + \tau_n) x_2 x_n \\ &+ \cdots \\ &+\tau_n^2 x_n^2. \end{aligned}$$

The Hessian matrix of w is,

$$H = \begin{bmatrix} \tau_1 + \tau_1 & \tau_1 + \tau_2 & \cdots & \tau_1 + \tau_{n-1} & \tau_1 + \tau_n \\ \tau_2 + \tau_1 & \tau_2 + \tau_2 & \cdots & \tau_2 + \tau_{n-1} & \tau_2 + \tau_n \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \tau_{n-1} + \tau_1 & \tau_{n-1} + \tau_2 & \cdots & \tau_{n-1} + \tau_{n-1} & \tau_{n-1} + \tau_n \\ \tau_n + \tau_1 & \tau_n + \tau_2 & \cdots & \tau_n + \tau_{n-1} & \tau_n + \tau_n \end{bmatrix}.$$

We now check the eigenvalues of H. Let t and e be the column vectors $(\tau_1, \tau_2, \ldots, \tau_n)^T$ and $(1, 1, \ldots, 1)^T$, respectively, both of length n. Then, each column of H is of the form $t + \tau_i e$, where i is the column number. Each row i of H is also of the form $(t + \tau_i e)^T$ as H is symmetric. Since we have assumed that not all τ_i's are equal, the vectors t and e are independent and the column space of H is the span of t and e. As a result, H is of dimension 2, which indicates that H has exactly 2 non-zero eigenvalues.

If v is an eigenvector of H, then it is a linear combination of t and e. One can easily verify that e is not an eigenvector, since we have assumed not all τ_i's are equal (See Lemma B.3 for proof by contradiction). Therefore, we can write $v = t + ce$ for some scalar c.

According to the definition of eigenvector, the equation $Hv = \lambda v$ should hold, where λ is the eigenvalue corresponding to the eigenvector v. Thus, for each row $i \in \{1, 2, 3, \ldots, n\}$ of H the following equation holds,

$$(t + \tau_i e)^T (t + ce) = \lambda(\tau_i + c).$$

Solving for c yields the following equation,

$$c = \frac{\sum_{j=1}^{n} \tau_j^2 + \tau_i \sum_{j=1}^{n} \tau_j - \lambda \tau_i}{\lambda - \sum_{j=1}^{n} \tau_j - n\tau_i}.$$

The above equation holds for every two rows k and l of matrix H, i.e.,

$$c = \frac{\sum_{j=1}^{n} \tau_j^2 + \tau_k \sum_{j=1}^{n} \tau_j - \lambda \tau_k}{\lambda - \sum_{j=1}^{n} \tau_j - n\tau_k} \text{ and } c = \frac{\sum_{j=1}^{n} \tau_j^2 + \tau_l \sum_{j=1}^{n} \tau_j - \lambda \tau_l}{\lambda - \sum_{j=1}^{n} \tau_j - n\tau_l}.$$

Since c is a constant, we can equate the right-hand sides of the above equations for rows l and k, where $l \neq k$,

$$c = \frac{\sum_{j=1}^{n} \tau_j^2 + \tau_k \sum_{j=1}^{n} \tau_j - \lambda \tau_k}{\lambda - \sum_{j=1}^{n} \tau_j - n\tau_k} = \frac{\sum_{j=1}^{n} \tau_j^2 + \tau_l \sum_{j=1}^{n} \tau_j - \lambda \tau_l}{\lambda - \sum_{j=1}^{n} \tau_j - n\tau_l}.$$

Solving for λ will result in,

$$-(\tau_k - \tau_l)\lambda^2 + 2\lambda(\tau_k - \tau_l) \sum_{j=1}^{n} \tau_j + n(\tau_k - \tau_l) \sum_{j=1}^{n} \tau_j^2 - (\tau_k - \tau_l)(\sum_{j=1}^{n} \tau_j)^2 = 0,$$

and since we assumed not all τ_i's are equal, the factor $(\tau_k - \tau_l)$ is not zero for at least for two columns,

$$-\lambda^2 + 2\lambda \sum_{j=1}^{n} \tau_j + n \sum_{j=1}^{n} \tau_j^2 - (\sum_{j=1}^{n} \tau_j)^2 = 0.$$

Solving for λ we then get,

$$\lambda = \sum_{j=1}^{n} \tau_j \pm \sqrt{n \sum_{j=1}^{n} \tau_j^2} = e^T t \pm \sqrt{n}\|t\|,$$

where $\|.\|$ is the norm operator. Since $e^T t$ is nonnegative, the Cauchy-Schwarz inequality, $(e^T t)^2 \leq \|e\|^2 \|t\|^2$, reduces to,

$$e^T t \leq \sqrt{n}\|t\|.$$

Knowing that the magnitude of e is $\sqrt{n}$ and given the assumption that the magnitude of t is greater than zero (we assumed that $\exists i : \tau_i > 0$), the two sides are equal if and only if vectors t and e are linearly dependent (i.e., they are parallel), which means that all τ_i's are equal. This contradicts our assumption. As a result, the smaller eigenvalue is always negative while the bigger one is always positive. Consequently, the Hessian matrix is indefinite and the bilinear term is non-convex. □

Lemma B.3. *Vector $e = (1, 1, \ldots, 1)^T$ cannot be an eigenvector for*

$$H = \begin{bmatrix} \tau_1 + \tau_1 & \tau_1 + \tau_2 & \cdots & \tau_1 + \tau_{n-1} & \tau_1 + \tau_n \\ \tau_2 + \tau_1 & \tau_2 + \tau_2 & \cdots & \tau_2 + \tau_{n-1} & \tau_2 + \tau_n \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \tau_{n-1} + \tau_1 & \tau_{n-1} + \tau_2 & \cdots & \tau_{n-1} + \tau_{n-1} & \tau_{n-1} + \tau_n \\ \tau_n + \tau_1 & \tau_n + \tau_2 & \cdots & \tau_n + \tau_{n-1} & \tau_n + \tau_n \end{bmatrix}.$$

Proof. Assume that e is an eigenvector for H and thus holds in $He = \lambda e$. Substitute H and e in the left hand-side of the equation,

$$He = \begin{bmatrix} \tau_1 + \tau_1 & \tau_1 + \tau_2 & \cdots & \tau_1 + \tau_{n-1} & \tau_1 + \tau_n \\ \tau_2 + \tau_1 & \tau_2 + \tau_2 & \cdots & \tau_2 + \tau_{n-1} & \tau_2 + \tau_n \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \tau_{n-1} + \tau_1 & \tau_{n-1} + \tau_2 & \cdots & \tau_{n-1} + \tau_{n-1} & \tau_{n-1} + \tau_n \\ \tau_n + \tau_1 & \tau_n + \tau_2 & \cdots & \tau_n + \tau_{n-1} & \tau_n + \tau_n \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} n\tau_1 + \sum_{j=1}^{n} \tau_j \\ n\tau_2 + \sum_{j=1}^{n} \tau_j \\ \vdots \\ n\tau_n + \sum_{j=1}^{n} \tau_j \end{bmatrix}.$$

There is no eigenvalue λ , for which the equality $He = \lambda e$ holds unless $\tau_k = \tau_l \quad \forall k, l \in \{1, 2, \ldots, n\}$. This contradicts our assumption. $\square$

C Proof of Proposition 1.3.3: Pseudo-Concavity of SL_{tot}

Similar to the proof in Appendix B, to show the that $SL(\lambda, r)$ is not jointly pseudo-concave in $\{\lambda_j\}$ and $\{r_j\}$ even in the special case with one single server at each group and an acceptable waiting time of zero, it suffices to examine the pseudo-convexity of the bilinear term $w_j = \lambda_j r_j$ in Equation (B.3) in $\{x_{ij}\}$. Lemma C.1 proves by a counter example that the bilinear term is not pseudo-convex in $\{x_{ij}\}$.

Lemma C.1. *For a given group j, if $\exists k, l \in FI_j$ such that $\tau_{kj} \neq \tau_{lj}$ and $\tau_{kj}, \tau_{lj} > 0$, the bilinear term $w_j = \lambda_j r_j = \sum_{i \in FI_j} x_{ij} \sum_{i \in FI_j} \tau_{ij} x_{ij}$ is not pseudo-convex in x_{ij}.*

Proof. We suppress index j in w_j and assume that the set FI_j is of length n. Then, the bilinear term can be expanded in as function of $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ as follows,

$$\begin{aligned} w(\mathbf{x}) = \sum_{i=1}^{n} x_i \sum_{i=1}^{n} \tau_i x_i = \tau_1 x_1^2 \quad &+(\tau_1 + \tau_2) x_1 x_2 + \cdots \quad +(\tau_1 + \tau_n) x_1 x_n \\ &+(\tau_2 + \tau_3) x_2 x_3 + \cdots \quad +(\tau_2 + \tau_n) x_2 x_n \\ &+ \cdots \\ &+\tau_n^2 x_n^2. \end{aligned}$$

The first gradient of $w(\mathbf{x})$ at $\bar{\mathbf{x}} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$ is,

$$\nabla w(\bar{\mathbf{x}}) = \begin{bmatrix} (\tau_1+\tau_1)\bar{x}_1 + (\tau_1+\tau_2)\bar{x}_2 + \cdots + (\tau_1+\tau_{n-1})\bar{x}_{n-1} + (\tau_1+\tau_n)\bar{x}_n \\ (\tau_2+\tau_1)\bar{x}_1 + (\tau_2+\tau_2)\bar{x}_2 + \cdots + (\tau_2+\tau_{n-1})\bar{x}_{n-1} + (\tau_2+\tau_n)\bar{x}_n \\ \vdots \\ (\tau_{n-1}+\tau_1)\bar{x}_1 + (\tau_{n-1}+\tau_2)\bar{x}_2 + \cdots + (\tau_{n-1}+\tau_{n-1})\bar{x}_{n-1} + (\tau_{n-1}+\tau_n)\bar{x}_n \\ (\tau_n+\tau_1)\bar{x}_1 + (\tau_n+\tau_2)\bar{x}_2 + \cdots + (\tau_n+\tau_{n-1})\bar{x}_{n-1} + (\tau_n+\tau_n)\bar{x}_n \end{bmatrix}$$

$$= \begin{bmatrix} \tau_1 \sum_{i=1}^{n} \bar{x}_i + \sum_{i=1}^{n} \tau_i \bar{x}_i \\ \tau_2 \sum_{i=1}^{n} \bar{x}_i + \sum_{i=1}^{n} \tau_i \bar{x}_i \\ \vdots \\ \tau_{n-1} \sum_{i=1}^{n} \bar{x}_i + \sum_{i=1}^{n} \tau_i \bar{x}_i \\ \tau_n \sum_{i=1}^{n} \bar{x}_i + \sum_{i=1}^{n} \tau_i \bar{x}_i \end{bmatrix}.$$

Let $\bar{\mathbf{y}} = (\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_n)$ be a vector similar to $\bar{\mathbf{x}} = (\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n)$, then, for pseudo-convexity one should prove that if $\nabla w(\bar{\mathbf{x}})(\bar{\mathbf{y}} - \bar{\mathbf{x}}) \geq 0$, then $w(\bar{\mathbf{y}}) \geq w(\bar{\mathbf{x}})$. The condition $\nabla w(\bar{\mathbf{x}})(\bar{\mathbf{y}} - \bar{\mathbf{x}}) \geq 0$ becomes,

$$\begin{aligned} &(\bar{y}_1 - \bar{x}_1)(\tau_1 \sum_{i=1}^{n} \bar{x}_i + \sum_{i=1}^{n} \tau_i \bar{x}_i) + (\bar{y}_2 - \bar{x}_2)(\tau_2 \sum_{i=1}^{n} \bar{x}_i + \sum_{i=1}^{n} \tau_i \bar{x}_i) + \ldots \\ &\quad + (\bar{y}_{n-1} - \bar{x}_{n-1})(\tau_{n-1} \sum_{i=1}^{n} \bar{x}_i + \sum_{i=1}^{n} \tau_i \bar{x}_i) + (\bar{y}_n - \bar{x}_n)(\tau_n \sum_{i=1}^{n} \bar{x}_i + \sum_{i=1}^{n} \tau_i \bar{x}_i) \\ &= \sum_{i=1}^{n} \tau_i \bar{x}_i (\sum_{i=1}^{n} \bar{y}_i - \sum_{i=1}^{n} \bar{x}_i) + \sum_{i=1}^{n} \bar{x}_i \sum_{i=1}^{n} \tau_i (\bar{y}_i - \bar{x}_i) \geq 0. \end{aligned}$$

Now, consider the following instance with $(\bar{x}_1, \bar{x}_2) = (2, 1)$, $(\bar{y}_1, \bar{y}_2) = (1, 3)$ and $(\tau_1, \tau_2) = (10, 1)$. The above condition becomes $21(4-1) + 3(-12) = 27 \geq 0$. However, $w(\bar{\mathbf{y}}) = 52 \leq$

$w(\bar{\mathbf{x}}) = 63$. □

D Proof of Proposition 1.4.1: Relaxed Problem Convexity

Lemmas D.1 and D.2 prove that SL_{tot}^{RL} is an upper bound for SL_{tot} and it is concave in λ_j, respectively. □

Lemma D.1. *The function SL_{tot}^{RL} is an upper bound for SL_{tot}.*

Proof. Lee & Cohen (1983) have shown that Erlang-C formula $C(k, r)$ is strictly increasing in r. Since $r = \tau\lambda$ and $\lambda \geq 0$, $C(k, \tau\lambda)$ is increasing in τ. Also, $e^{-(\frac{k-r}{\tau})T} = e^{-(\frac{k}{\tau}-\lambda)T}$ is increasing in τ. Thus, $SL(\lambda, \tau\lambda) = 1 - C(k, \tau\lambda)e^{-(\frac{k}{\tau}-\lambda)T}$ is decreasing in τ. Since we choose τ_j^{FIX} such that $\tau_j^{FIX} \leq \tau_j$, then,

$$SL(\lambda_j, \tau_j^{FIX}\lambda_j) \geq SL(\lambda_j, r_j) \quad \forall j \in J. \tag{D.6}$$

Because $\lambda_j \geq 0 \ \forall j \in J$,

$$\lambda_j SL(\lambda_j, \tau_j^{FIX}\lambda_j) \geq \lambda_j SL(\lambda_j, r_j) \quad \forall j \in J.$$

Since sum preserves the monotonicity,

$$SL_{tot}^{RL} = \sum_{j \in J} \lambda_j SL(\lambda_j, \tau_j^{FIX}\lambda_j) \geq \sum_{j \in J} \lambda_j SL(\lambda_j, r_j) = SL_{tot}.$$

□

Lemma D.2. *The function SL_{tot}^{RL} is concave in λ_j with fixed τ_j.*

Proof. Lemma D.3 proves that Erlang-C function $C(k, \tau\lambda)$ is nonnegative nondecreasing convex in λ with fixed τ. Also, $e^{-(\frac{k}{\tau}-\lambda)T}$ is nonnegative nondecreasing convex in λ with fixed τ. Thus, the product $C(k, \tau\lambda)e^{-(\frac{k}{\tau}-\lambda)T}$ is nonnegative nondecreasing convex in λ, and

$$SL(\lambda, \tau\lambda) = 1 - C(k, \tau\lambda)e^{-(\frac{k}{\tau}-\lambda)T},$$

is nonnegative nonincreasing concave in λ with fixed τ. The second derivative of the term $\lambda_j SL(\lambda_j, \tau_j\lambda_j)$ with respect to λ_j is (for brevity we drop the index j),

$$\frac{\partial^2 \lambda SL(\lambda, \tau\lambda)}{\partial\lambda^2} = 2\frac{\partial SL(\lambda, \tau\lambda)}{\partial\lambda} + \lambda\frac{\partial^2 SL(\lambda, \tau\lambda)}{\partial\lambda^2},$$

which is non-positive since λ is nonnegative and $SL(\lambda, \tau\lambda)$ is nonincreasing concave in λ. Finally, since sum preserves concavity, $SL^{RL}_{tot} = \sum_{j\in J} \lambda_j SL(\lambda_j, \tau^{FIX}_j \lambda_j)$ is concave in $\{\lambda_j\}$ for any fixed $\tau_j = \tau^{FIX}_j$. □

Lemma D.3. *Erlang-C function $C(k, \tau\lambda)$ is nonnegative nondecreasing convex in arrival rate λ with fixed τ.*

Proof. Lee & Cohen (1983) have proved that Erlang-C function $C(k, r)$ is nonnegative nondecreasing convex in r. Since $\lambda \geq 0$, $C(k, \tau\lambda)$ is nonnegative nondecreasing convex in λ with fixed τ. □

E Subproblem

After solving the relaxed problem, either (RL) or $(RL2)$, and finding $\{x_{ij}\}$, $\{r_j\}$ and $\{\lambda_j\}$, we can seek a better solution by solving a *subproblem* (SP_r).

Specifically, we fix the workloads $\left\{r^{RL}_j\right\}$ obtained from the relaxed problem solution and re-optimize $\{x_{ij}\}$ and $\{\lambda_j\}$. Proposition E.1 proves that the corresponding subproblem is a

convex program. Given the fixed values for a subset of variables, i.e., $\left\{r_j^{RL}\right\}$, we optimize over the remaining variables $\{x_{ij}\}$ and $\{\lambda_j\}$. The subproblem (SP_r) is as follows,

$$(SP_r) \quad \max \quad \sum_{j\in J} \lambda_j SL(\lambda_j, r_j^{RL}) \tag{E.7}$$

$$\text{s.t.} \quad (1.2),\ (1.3),\ (1.10) \text{ and } (1.12). \tag{E.8}$$

Note that since constraint (1.9) is satisfied in the relaxed problem, we don't require it in (SP_r). After solving (SP_r), the optimality gap between the relaxed problem and the subproblem is calculated as,

$$Gap_{sp} = \sum_{j\in J} \lambda_j^{SP} SL(\lambda_j^{SP}, r_j^{RL}) - \sum_{j\in J} \lambda_j^{RL} SL(\lambda_j^{RL}, r_j^{RL}),$$

where we obtain $\left\{\lambda_j^{SP}\right\}$ from the subproblem solution and $\left\{r_j^{RL}\right\}$ from the relaxed problem solution. This gap is at least as good as the gap found by the relaxed problem.

One might want to skip solving the subproblem. Our experiments show that subproblem can often find a better solution. Moreover, (SP_r) is not an expensive problem since it is a continuous convex problem with no integer or binary variable, while $(RL2)$ is a mixed-integer nonlinear problem.

Proposition E.1. *The subproblem (SP_r) is convex in $\{\lambda_j\}$ and $\{x_{ij}\}$ with $\{r_j^{RL}\}$ being fixed.*

Proof. The delay probability Erlang-C function $C(k_j, r_j^{RL})$ becomes a parameter with fixed r_j^{RL}. Substituting for $\tau_j = r_j^{RL}/\lambda_j$, the conditional delay formula can be reformulated as,

$$exp(-(\frac{k_j - r_j^{RL}}{\tau_j})T) = exp(\frac{r_j^{RL} - k_j}{r_j^{RL}} T\lambda_j),$$

which is nondecreasing convex in λ_j. Thus, $SL(\lambda_j, r_j^{RL})$ as defined in (1.6) is nonincreasing

concave in λ_j. Since sum preserves monotonicity and concavity, the objective function of the subproblem (SP_r) as defined in (E.7) is concave in $\{\lambda_j\}$. All the constraints of (SP_r) are linear in $\{x_{ij}\}$ and $\{\lambda_j\}$. Thus, the subproblem (SP_r) is a convex problem in $\{\lambda_j\}$ and $\{x_{ij}\}$. □

F FSE Algorithm: Subroutines for Adding and Removing Boundaries

In Table F.1 we explain the FSE subroutine for adding new boundaries. Tables F.2 and F.3 explain subroutines for choosing group to remove one or boundaries from as well as specific boundary or boundaries to be removed, respectively.

G Experiment results

Tables G.4 and G.6 list the results of our experiments on a total of 20 instances (6 planar and 14 non-planar instances). The desired optimal percentage gap (see 1.33 for the definition of Gap^{pct}) used to terminate the solution methods is 1% in Table G.4 and 5% in Table G.6. In these tables we compare the performance of the solution methods in terms of their respective solution quality measured by the percentage gap Gap^{pct} as well as the time they need to find that solution.

Table F.1: FSE: FSE^{ADD}

1. Inputs and Required Parameters

- θ^{MAX} and θ^{MIN} in equation (1.27).

- $\{T_{j,t}\}$= ordered list of slice boundaries at iteration t.

- $\{A_{j,t-1}\}$= set of actual service times generated from iteration 1 to iteration $t-1$. (For $t=1$, $A_{j,0} \leftarrow \{\}\ \forall j$)

2. Output

- $\{T^{add}_{j,t+1}\}$= ordered list of boundaries with the newly generated boundary for iteration $t+1$.

- $\{A_{j,t}\}$= set of actual service times generated from iteration 1 to iteration $t+1$.

3. Main Algorithm

- For each group j,

  • If $\lambda_j = 0$,

    ◦ $\tau^{ACTUAL}_{j,t} \leftarrow 0$.

  else,

    ◦ $\tau^{ACTUAL}_{j,t} \leftarrow r_j/\lambda_j$.

    ◦ Construct $A_{j,t}$: $A_{j,t} \leftarrow A_{j,t-1} + \tau^{ACTUAL}_{j,t}$.

  • Store the index of the slice that the solution (λ_j, r_j) is in:

    $n^* \leftarrow \{n | \tau_{j,n} \leq \tau^{ACTUAL}_{j,t} < \tau_{j,n+1}\}$.

  • Determine δ according to (1.26).

  • Determine θ according to (1.27).

  • Create a new boundary τ^{NEW}_j between the current lower boundary τ_{j,n^*} and the actual mean service time $\tau^{ACTUAL}_{j,t}$: $\tau^{NEW}_j \leftarrow \theta\tau^{ACTUAL}_{j,t} + (1-\theta)\tau_{j,n^*}$.

  • Construct $T^{add}_{j,t+1}$: $T^{add}_{j,t+1} \leftarrow \left\{\tau_{j,n} \in T_{j,t} : \tau_{j,n} \leq \tau^{ACTUAL}_{j,t}, n \in \{1,\ldots,n^*\}\right\} + \tau^{NEW}_j + \left\{\tau_{j,n} \in T_{j,t} : \tau_{j,n} > \tau^{ACTUAL}_{j,t}, n \in \{n^*+1,\ldots,m_j+1\}\right\}$.

Table F.2: FSE: FSE_{gr}^{REM}

1. Inputs and Required Parameters

- ψ^{tot} = maximum total number of boundaries to keep across all group.

- ψ^{gr} = maximum number of boundaries to keep for each group.

- $\{T_{j,t}^{add}\}$= ordered list of boundaries with the newly generated boundary for iteration t (See Table F.1).

- $\{A_{j,t}\}$= set of actual service times generated from iteration 1 to iteration t.

2. Output

- $\{T_{j,t+1}\}$= ordered list of boundaries for iteration $t+1$.

3. Main Algorithm

- For each group j, initialize $T_{j,t}^{rem}$, ordered list of boundaries for iteration t after removing boundaries: $T_{j,t}^{keep} \leftarrow T_{j,t}^{add}\ \forall j$.

- For each group j:

  • If $\psi^{gr} > \left|T_{j,t}^{keep}\right|$, then remove one boundary from group j and update $T_{j,t}^{keep}$ (See FSE_{bnd}^{REM} in Table F.3).

- If $\psi^{tot} > \sum_j \left|T_{j,t}^{keep}\right|$, then,

  • Use equation (1.28) to calculate Gap_j.

  • Sort groups from smallest to largest according to their Gap_j and store the result in the ordered list $Sort$.

- Repeat until $\psi^{tot} \leq \sum_j \left|T_{j,t}^{keep}\right|$:

  • From the top of the list, $Sort$, pick j_{top} and remove it from $Sort$.

  • If j_{top} has four or more boundaries, then remove two boundaries and update $T_{j_{top},t}^{keep}$ (See FSE_{bnd}^{REM} in Table F.3 and use $ToRemove = 2$);

  • else if j_{top} has exactly four boundaries, then remove one boundary and update $T_{j_{top},t}^{keep}$ (See FSE_{bnd}^{REM} in Table F.3 and use $ToRemove = 1$);

  • else, do not remove any boundary.

- Construct the boundary list for iteration $t+1$ and terminate: $T_{j,t+1} \leftarrow T_{j,t}^{keep}\ \forall j$.

Table F.3: FSE: FSE_{bnd}^{REM}

1. Inputs and Required Parameters

- $\{T_{j,t}^{add}\}$= ordered list of boundaries with the newly generated boundary for iteration t (See Table F.1).

- $\{A_{j,t}\}$= set of actual service times generated from iteration 1 to iteration t.

- $ToRemove$= Number of boundaries that should be removed.

2. Output

- $\{T_{j,t}^{keep}\}$= ordered list of boundaries for iteration t after removing $ToRemove$ boundaries.

3. Main Algorithm

Initialize $removed$: $removed \leftarrow 0$.

- *Scoring the boundaries of* $T_{j,t}^{add}$*:*

  - Reset scores for all the boundaries in $T_{j,t}^{add}$: $S(\tau_{j,n}) \leftarrow 0, \ \forall n$.

  - For each boundary $\tau_{j,n} \in T_{j,t}^{add}$ calculate its score (See equation (1.29)).

- Initialize a list for boundaries that we keep:

  $T_{j,t}^{keep} \leftarrow \{\tau_j^{MIN}, \tau_j^{NEW}, \tau_j^{MAX}\}$.

- Initialize a list for boundaries that we can remove: $T_j^{temp} \leftarrow T_{j,t}^{add} \setminus T_j^{keep}$.

- Form a set for boundaries with a score of zero: $T_j^{zero} \leftarrow \{\tau_{j,n} \in T_j^{temp} | S(\tau_{j,n}) = 0\}$.

- While $|T_j^{zero}| > 0$ and $removed < ToRemove$:

  - Randomly choose one boundary called τ from T_j^{zero}.

  - Remove the selected boundary, τ, from T_j^{zero} and T_j^{temp}:

  $T_j^{temp} \leftarrow \{T_j^{temp}\} \setminus \tau$ and $T_j^{keep} \leftarrow \{T_j^{keep}\} \setminus \tau$

  - Update $removed$: $removed \leftarrow reomoved + 1$.

- While $|T_{j,t}^{keep}| \leq |T_{j,t}^{add}| - ToRemove + removed$:

  - Calculate cumulative scores for boundaries in T_j^{temp} (See equation (1.30)).

  - Generate a uniform random number $rand$ (See equation (1.31)).

  - Select boundary $\tau_{j,n} \in T_j^{temp}$ such that $S^{cum}(\tau_{j,n-1}) \leq rand < S^{cum}(\tau_{j,n})$.

  - Remove the selected boundary from T_j^{temp} and add it to $T_{j,t}^{keep}$ (See equation (1.32)).

- Return $T_{j,t}^{keep}$ and terminate.

Table G.4: Experiment results for the percentage gap of 1%.

Instance					KNITRO		BARON		FSE		SQE		FE-even		FE-add	
No.	Type[1]	$\|I\|$	$\|J\|$	Density	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]
1	P	14	9	0.17	Inf.[3]	0	66.51	43282	0.10	66	0.37	16	26.94	738	0.11	70
2	P	18	13	0.10	Inf.	0	Inf.	43257	0.66	7	0.67	1	14.81	279	0.69	8
3	P	20	19	0.08	Inf.	0	Inf.	43499	0.52	14	0.13	7	38.58	0	0.52	14
4	P	39	10	0.19	Inf.	0	Inf.	43225	0.21	215	0.67	28	45.18	3	0.21	263
5	P	37	12	0.16	Inf.	0	Inf.	43201	0.50	3314	0.31	44124	68.34	139	47.34	740
6	P	45	13	0.14	Inf.	0	Inf.	43364	0.94	38722	0.75	43769	65.01	2	2.60	1236
7	NP	4	5	1.00	Inf.	0	17.51	43241	0.87	0	0.64	0	7.85	0	0.87	0
8	NP	8	5	1.00	Inf.	0	5.36	43296	0.77	4	0.67	2	73.71	0	4.48	1
9	NP	10	5	1.00	Inf.	0	28.99	43215	0.84	3512	12.47	43337	94.55	0	6.55	95
10	NP	10	7	1.00	Inf.	0	Inf.	43215	0.51	14	0.64	7	29.05	0	0.51	14

[1] Time is measured in seconds.
[2] Instance type is Non-planar (NP) or Planar (P).
[3] The solver is stuck in a local optimum with a very large optimality gap denoted by infinity.

Table G.5: Experiment results for the percentage gap of 1% (Continued).

Instance					KNITRO		BARON		FSE		SQE		FE-even		FE-add	
No.	Type[1]	$\|I\|$	$\|J\|$	Density	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]
11	NP	20	10	1.00	Inf.	0	13.69	43238	0.89	21	0.79	139	3.47	1	5.21	1
12	NP	30	12	1.00	Inf.	0	Inf.	43223	0.77	24349	13.56	43070	44.81	1	29.20	4
13	NP	50	10	1.00	Inf.	0	Inf.	43285	0.98	38722	5.82	43769	93.22	2	16.35	1236
14	NP	8	5	0.73	Inf.	0	Inf.	43223	0.87	130	9.90	43175	91.11	0	5.94	43
15	NP	12	5	0.72	Inf.	0	Inf.	43381	0.91	83	0.94	111	76.52	0	1.84	52
16	NP	15	6	0.79	Inf.	0	Inf.	43328	0.74	1096	4.46	43209	85.10	1	2.45	331
17	NP	20	6	0.72	Inf.	0	Inf.	43346	0.98	84	0.46	60	71.10	0	0.98	65
18	NP	15	7	0.63	Inf.	0	Inf.	43290	0.50	1494	3.55	43426	56.38	1	0.50	1494
19	NP	25	8	0.53	Inf.	0	Inf.	43279	0.35	801	0.54	1460	61.56	3	1.97	245
20	NP	30	10	0.40	Inf.	0	195.91	43302	0.82	8548	3.40	43419	80.27	3	25.92	128

[1] Time is measured in seconds.
[2] Instance type is Non-planar (NP) or Planar (P).
[3] The solver is stuck in a local optimum with a very large optimality gap denoted by infinity.

Table G.6: Experiment results for the percentage gap of 5%.

Instance					KNITRO		BARON		FSE		SQE		FE-even		FE-add	
No.	Type[1]	$\|I\|$	$\|J\|$	Density	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]
1	P	14	9	0.17	Inf.[3]	0	66.51	43282	1.29	0	3.51	0	26.94	738	1.29	0
2	P	18	13	0.10	Inf.	0	Inf.	43257	2.13	1	1.96	1	14.81	279	2.13	0
3	P	20	19	0.08	Inf.	0	Inf.	43499	1.06	3	2.06	3	38.58	0	1.06	3
4	P	39	10	0.19	Inf.	0	Inf.	43225	2.14	7	1.99	18	45.18	3	2.14	7
5	P	37	12	0.16	Inf.	0	Inf.	43201	0.67	1593	1.40	102	68.34	139	47.34	740
6	P	45	13	0.14	Inf.	0	Inf.	43364	4.34	280	4.12	44	65.01	2	4.34	280
7	NP	4	5	1.00	Inf.	0	17.51	43241	0.87	0	1.55	0	7.85	0	0.87	0
8	NP	8	5	1.00	Inf.	0	5.36	43296	4.48	1	1.01	1	73.71	0	4.48	1
9	NP	10	5	1.00	Inf.	0	28.99	43215	3.32	225	12.47	43337	94.55	0	6.55	95
10	NP	10	7	1.00	Inf.	0	Inf.	43215	3.46	1	1.66	4	29.05	0	3.46	1

[1] Time is measured in seconds.
[2] Instance type is Non-planar (NP) or Planar (P).
[3] The solver is stuck in a local optimum with a very large optimality gap denoted by infinity.

Table G.7: Experiment results for the percentage gap of 5% (Continued).

Instance					KNITRO		BARON		FSE		SQE		FE-even		FE-add	
No.	Type[1]	$\|I\|$	$\|J\|$	Density	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]	Gap^{pct}	Time[2]
11	NP	20	10	1.00	Inf.	0	13.69	43238	2.62	7	2.08	9	3.47	1	5.21	1
12	NP	30	12	1.00	Inf.	0	Inf.	43223	2.37	486	13.56	43070	44.81	1	29.20	4
13	NP	50	10	1.00	Inf.	0	Inf.	43285	2.23	8096	5.82	43769	93.22	2	16.35	1236
14	NP	8	5	0.73	Inf.	0	Inf.	43223	2.40	112	9.90	43175	91.11	0	5.94	43
15	NP	12	5	0.72	Inf.	0	Inf.	43381	2.17	22	1.04	33	76.52	0	2.17	22
16	NP	15	6	0.79	Inf.	0	Inf.	43328	2.90	92	4.46	19519	85.10	1	2.90	93
17	NP	20	6	0.72	Inf.	0	Inf.	43346	2.52	17	2.37	2	71.10	0	2.52	17
18	NP	15	7	0.63	Inf.	0	Inf.	43290	0.50	1494	4.01	228	56.38	1	0.50	1494
19	NP	25	8	0.53	Inf.	0	Inf.	43279	1.97	243	4.08	12	61.56	3	1.97	245
20	NP	30	10	0.40	Inf.	0	195.91	43302	3.13	347	3.42	4961	80.27	3	25.92	128

[1] Time is measured in seconds.
[2] Instance type is Non-planar (NP) or Planar (P).
[3] The solver is stuck in a local optimum with a very large optimality gap denoted by infinity.

H Math Programs $(RL3)$ and $(RS3)$

In the following we introduce relaxed problem $(RL3)$.

$$
\begin{aligned}
(RL3) \quad \max \quad & \sum_{j \in J} \lambda_j \\
\text{s.t.} \quad & \text{Constraints (1.2), (1.3), (1.9), (1.10), (1.11), (1.12), (1.19), (1.20),} \\
& \text{(1.21), (1.22), (1.23), (1.24), (1.25), (1.23),(1.24), (1.25),} \\
& \sum_{j \in J} \sum_{n \in N_j} \lambda_{jn} SL(\lambda_{jn}, \tau_{j,n} \lambda_{jn}) \geq \alpha_{avg}^{SL} \sum_{j \in J} \lambda_j,
\end{aligned}
$$

In problem $(RL3)$ we underestimate the workload r_j by $\tau_{j,n}\lambda_{j,n}$ in slice n. On the other hand, in problem $(RS3)$ we overestimate the workload r_j by $\tau_{j,n+1}\lambda_{j,n}$ in slice n.

$$
\begin{aligned}
(RS3) \quad \max \quad & \sum_{j \in J} \lambda_j \\
\text{s.t.} \quad & \text{Constraints (1.2), (1.3), (1.9), (1.10), (1.11), (1.12), (1.19), (1.20),} \\
& \text{(1.21), (1.22), (1.23), (1.24), (1.25), (1.23),(1.24), (1.25),} \\
& \sum_{j \in J} \sum_{n \in N_j} \lambda_{jn} SL(\lambda_{jn}, \tau_{j,n+1} \lambda_{jn}) \geq \alpha_{avg}^{SL} \sum_{j \in J} \lambda_j,
\end{aligned}
$$

I Proof of Proposition 2.3.1: Binding Coverage for EW_{tot}

Assume that for the optimal solution $\left\{x_{ij}^*\right\}$ with the corresponding arrival rate $\left\{\lambda_j^*\right\}$ and workload $\left\{r_j^*\right\}$ the minimum coverage imposed in Constraint (2.12) is not binding, i.e.,

$$\sum_{j \in J} \lambda_j^* > CF \sum_{i \in I} d_i.$$

Construct binding solution $\{\bar{x}_{ij}\}$ with corresponding $\{\bar{\lambda}_j\}$ and $\{\bar{r}_j\}$ as follows,

$$\bar{x}_{ij} = \beta x^*_{ij} \quad \forall (i,j) \in F,$$

where,

$$\beta = \frac{CF \sum_{i \in I} d_i}{\sum_{j \in J} \lambda^*_j}.$$

Note that $0 \leq \beta < 1$ and $\sum_{j \in J} \bar{\lambda}_j = CF \sum_{i \in I} d_i$. Then,

$$\bar{\lambda}_j = \sum_{i \in I} \bar{x}_{ij} = \sum_{i \in I} \beta x^*_{ij} = \beta \lambda^*_j < \lambda^*_j \quad \forall j \in J, \tag{I.9}$$

$$\bar{r}_j = \sum_{i \in I} \tau_{ij} \bar{x}_{ij} = \sum_{i \in I} \tau_{ij} \beta x^*_{ij} = \beta r^*_j < r^*_j \quad \forall j \in J, \tag{I.10}$$

and

$$\bar{\tau}_j = \frac{\bar{r}_j}{\bar{\lambda}_j} = \frac{\beta r^*_j}{\beta \lambda^*_j} = \tau^*_j \quad \forall j \in J.$$

Lemma I.1 (which follows) prove that $EW(\lambda, r)$ is strictly increasing in λ and r with fixed $\tau > 0$. Thus,

$$EW(\bar{\lambda}_j, \bar{r}_j) < EW(\lambda^*_j, r^*_j) \quad \forall j \in J.$$

Using inequalities (I.9) and (I.10), the following inequality follows,

$$\begin{aligned}
\overline{EW}_{avg} &= \frac{\sum_{j \in J} \bar{\lambda}_j EW(\bar{\lambda}_j, \bar{r}_j)}{\sum_{j \in J} \bar{\lambda}_j} \\
&= \frac{\sum_{j \in J} \lambda^*_j EW(\bar{\lambda}_j, \bar{r}_j)}{\sum_{j \in J} \lambda^*_j} \\
&< \frac{\sum_{j \in J} \lambda^*_j EW(\lambda^*_j, r^*_j)}{\sum_{j \in J} \lambda^*_j} = EW^*_{avg},
\end{aligned}$$

which contradicts the optimality assumption for solution $\left\{x_{ij}^*\right\}$ with objective value EW_{avg}^*.

□

Lemma I.1. *Expected waiting time $EW(\lambda, r)$ is strictly increasing in λ and r with fixed $\tau > 0$.*

Proof. Lee & Cohen (1983) have proved that Erlang-C formula $EC(k, r)$ is strictly increasing in workload r. Also, $\tau/(k - r)$ is positive and strictly increasing in r with fixed τ. Thus, $EW(\lambda, r)$ as a product of two positive strictly increasing functions is positive and strictly increasing in r. Fixing $\tau > 0$ drives λ linear in $r = \tau\lambda$. Thus, $EW(\lambda, r)$ is jointly increasing in λ and r with fixed τ. □

J Proof of Proposition 2.3.2: Convexity of EW_{tot}

Proof. According to Little's law (i.e., *queue length = arrival rate × waiting time*), for each group j the corresponding term $\lambda_j EW(\lambda_j, r_j)$ is the expected queue length (denoted by $E[L]$) for that group, i.e.,

$$E[L_j] = \lambda_j EW(\lambda_j, r_j) = EC(k_j, r_j)[r_j/(k_j - r_j)],$$

where L_j is the random number of jobs in the group j's queue. Grassmann (1983) proved that expected queue length $E[L_j]$ is convex in traffic intensity ρ_j and since $r_j = \rho_j k_j$, $E[L_j]$ is convex in workload r_j with fixed k_j. Since sum preserves the convexity, $\sum_{j \in J} E[L_j] = \sum_{j \in J} \lambda_j EW(\lambda_j, r_j)$ is convex in r_j's. Note that $E[L_j]$ is a function of r_j and variables τ_j and λ_j do not appear in $E[L_j]$. □

K OptXRand Map and the Corresponding Static Routing Map

In the OptXRand policy, when an incident in block group i occurs it is randomly routed to $j \in FK_i$ according to the x_{ij}'s. We use x_{ij}'s to build new zones where each zone is either not covered or is assigned to exactly one fire station. In this new map there is no random assignment as opposed to the original OptXRand. We divide the block groups according to the x_{ij}'s to build a new map with new arrival rates. For example, let the demand at block group A be 2, i.e., $d_A = 2$. Also, assume that we route $x_{A1} = 0.8$ to station 1 and $x_{A2} = 1.2$ to station 2, i.e., we don't block jobs upon arrival. We divide block group A into two zones, B and C with areas proportional to $x_{A1}/(x_{A1} + x_{A2}) = 0.4$ and $x_{A2}/(x_{A1} + x_{A2}) = 0.6$. Zone B is entirely assigned to group 1 with the new arrival rate equal to $x_{B1} = x_{A1}$, zone C is entirely assigned to group 2 with the arrival rate equal to $x_{C2} = x_{A2}$, and $x_{B2} = x_{C1} = 0$. With this mapping the routings are predetermined and all the incidents in a zone are assigned to a fixed station with no randomization. Moreover, the new routing map has the same performance as its corresponding OptXRand policy, because the arrival rates to the fire stations are preserved under this new map.

We build a routing map for Irvine fire stations based on the OptXRand policy (See Figure 2.10). As depicted in Figure 2.10 block group i is linked to station $j \in FK_i$, if x_{ij} is strictly positive. There are multiple ways to split a block to multiple zones and Figure K.1 shows one static routing map corresponding to the map in Figure 2.10. In Figure K.1 each zone is linked to at most one station.

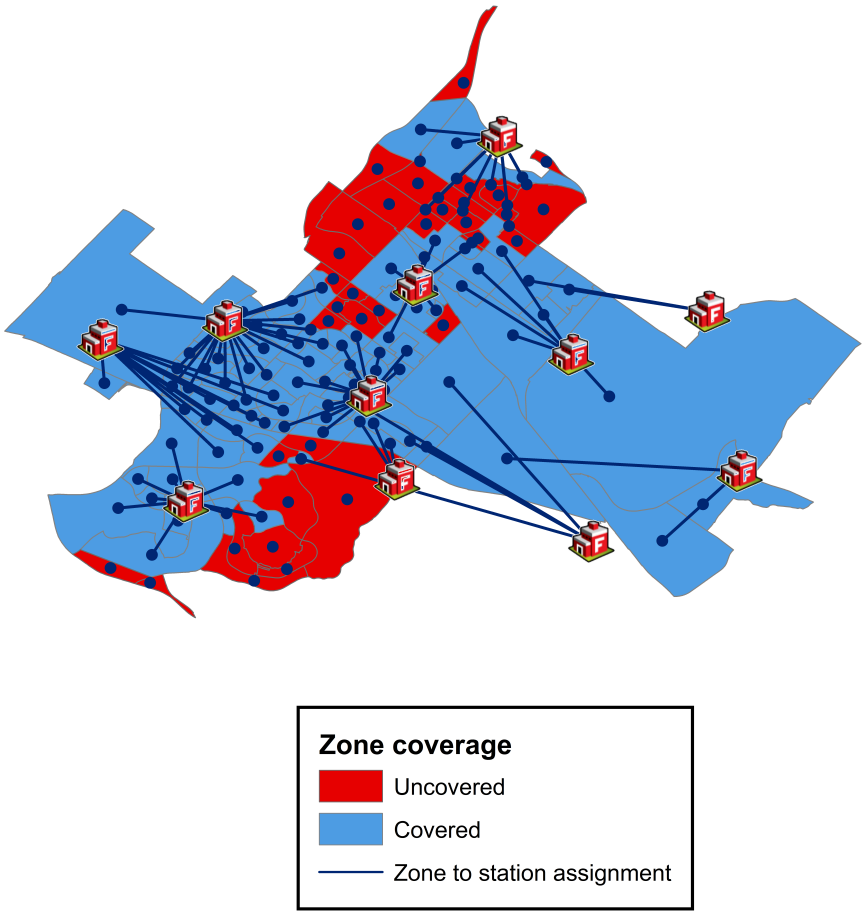


Figure K.1: Static routing map corresponding to the OptXRand map in Figure 2.10

L Simulation results

Tables L.8, L.9 and L.10 list simulation results. In these tables we compare the performance of routing policies in terms of the expected waiting time that jobs experience, namely, EW_{tot} or EW_{avg}. Notice that the optimal (or, minimum) EW_{avg} is equivalent to the optimal EW_{tot} since $EW_{avg} = \sum_{j \in J} \lambda_j EW_{tot}$, and the coverage constraint on $\sum_{j \in J} \lambda_j$ is binding at optimality (see proposition 2.3.1). We run simulations on a total of 20 instances: 14 non-planar instances (to resemble call center applications), and 6 planar instances (to resemble transportation systems).

Table L.8: Simulation results for test cases 1 to 15 (non-planar instances 1 to 5).
(**Bold face**: Best routing policy in the respective coverage factor.)

			FSF		FSFBlock		OptXRand		OptXOverflow		OptXOverflow-Block		FSF-OptXOverflow		FSFOptX-OverflowBlock	
Test Number	Instance	Coverage Factor	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}
1	1	0.7	6148.1	617.8	1313.5	155.9	5.2	0.6	6144.5	617.4	**1.8**	**0.2**	6160.2	619.2	1.9	0.2
2	1	0.75	6148.1	617.8	2484.2	285.4	12.1	1.3	6642.4	641.7	**4.3**	**0.5**	6593.5	636.7	4.8	0.5
3	1	0.8	6148.1	617.8	3479.6	388.0	44.8	4.5	6625.3	640.2	18.3	1.8	6599.1	637.6	**17.8**	**1.8**
4	2	0.3	14231.2	859.5	2101.1	187.3	6.3	0.5	14148.2	856.7	**4.7**	**0.4**	14243.9	860.3	5.6	0.4
5	2	0.35	14231.2	859.5	3952.2	317.3	27.2	1.8	14166.8	857.7	17.2	1.2	14240.6	860.0	**16.8**	**1.1**
6	2	0.4	14231.2	859.5	4341.7	367.4	601.5	36.5	14255.5	862.4	363.2	22.0	14232.4	859.3	**313.0**	**19.0**
7	3	0.35	41168.3	1237.4	97.6	3.8	9.5	0.4	41146.1	1236.8	9.6	0.4	41165.4	1237.2	**9.4**	**0.4**
8	3	0.4	41168.3	1237.4	1980.8	67.7	32.1	1.1	41136.4	1236.7	32.2	1.1	41155.4	1236.9	**23.0**	**0.8**
9	3	0.45	41168.3	1237.4	3010.7	109.2	424.7	13.0	41164.7	1237.4	417.4	12.7	41151.7	1236.6	**208.4**	**6.3**
10	4	0.9	2.0	0.1	0.2	0.0	0.8	0.0	28.2	1.1	0.3	0.0	1.5	0.1	**0.0**	**0.0**
11	4	0.95	2.0	0.1	0.8	0.0	1.3	0.1	3.9	0.2	0.8	0.0	2.0	0.1	**0.2**	**0.0**
12	4	1	2.0	0.1	2.0	0.1	2.4	0.1	**0.8**	**0.0**	**0.8**	**0.0**	0.9	0.0	0.9	0.0
13	5	0.9	2608.2	58.5	47.7	1.2	11.6	0.3	63.0	1.7	**2.1**	**0.1**	91.2	2.0	4.1	0.1
14	5	0.95	2608.2	58.5	1485.1	34.7	15.3	0.4	31.7	0.7	**4.5**	**0.1**	21.6	0.5	10.7	0.2
15	5	1	2608.2	58.5	2608.2	58.5	39.2	0.9	**13.8**	**0.3**	**13.8**	**0.3**	22.2	0.5	22.2	0.5

Table L.9: Simulation results for test cases 16 to 28 (non-planar instances 6 to 12).
(**Bold face**: Best routing policy in the respective coverage factor.)

			FSF		FSFBlock		OptXRand		OptXOverflow		OptXOverflow-Block		FSF-OptXOverflow		FSFOptX-OverflowBlock	
Test Number	Instance	Coverage Factor	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}
16	6	0.2	18044.5	471.0	2905.3	118.4	7.6	0.3	7815.8	229.0	**6.1**	**0.2**	13685.2	356.4	17.0	0.7
17	6	0.25	18044.5	471.0	6017.6	220.2	29.3	0.9	8955.3	243.2	**23.2**	**0.7**	13883.4	361.4	31.0	1.0
18	6	0.3	18044.5	471.0	6851.0	234.7	1097.1	28.6	14871.4	386.9	1213.3	31.7	14904.2	387.7	**586.2**	**15.3**
19	7	0.3	23054.0	1051.7	2260.1	135.8	9.3	0.5	12281.6	579.4	**5.4**	**0.3**	17035.0	763.4	8.7	0.5
20	7	0.35	23054.0	1051.7	3944.2	223.1	86.8	4.0	20789.6	926.9	64.9	3.0	20538.4	913.1	**42.1**	**1.9**
21	8	0.4	12148.0	659.1	3352.3	232.6	11.0	0.7	7175.8	408.3	7.3	0.5	7734.2	435.7	**6.8**	**0.4**
22	8	0.45	12148.0	659.1	4942.5	329.4	53.7	3.1	7134.9	406.1	**39.2**	**2.2**	7741.5	436.1	57.7	3.3
23	9	0.3	18784.3	819.5	4225.7	260.2	9.5	0.5	13892.1	611.4	**5.6**	**0.3**	18769.5	815.1	11.1	0.6
24	9	0.35	18784.3	819.5	5211.7	320.9	72.9	3.3	13560.0	597.1	**53.0**	**2.4**	13614.5	599.5	79.5	3.6
25	10	0.25	23951.4	1052.4	3825.7	217.0	19.1	0.9	23665.0	1013.3	11.4	0.6	25979.3	1107.1	**8.8**	**0.4**
26	10	0.3	23951.4	1052.4	5333.0	284.0	**768.9**	**32.8**	24300.8	1037.7	1130.9	48.5	25912.9	1104.5	944.1	40.2
27	11	0.2	19930.9	728.7	2982.9	149.3	13.5	0.6	14898.5	555.5	**10.8**	**0.5**	15852.7	586.2	11.2	0.5
28	12	0.3	28306.9	798.6	5491.4	214.6	44.2	1.4	27996.2	785.7	34.0	1.1	28060.5	787.7	**26.3**	**0.8**

Table L.10: Simulation results for test cases 29 to 42 (non-planar instances 13 and 14, and planar instances 15 to 20). (**Bold face**: Best routing policy in the respective coverage factor.)

			FSF		FSFBlock		OptXRand		OptXOverflow		OptXOverflow-Block		FSF-OptXOverflow		FSFOptX-OverflowBlock	
Test Number	Instance	Coverage Factor	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}	EW_{tot}	EW_{avg}
29	13	0.15	21411.8	804.9	2392.1	136.2	12.3	0.6	16967.4	669.5	**7.4**	**0.4**	21608.2	813.2	11.7	0.6
30	14	0.4	16645.7	455.3	46.0	1.6	15.4	0.5	13599.3	372.2	12.4	0.4	16321.2	444.8	**4.8**	**0.2**
31	14	0.45	16645.7	455.3	3276.3	104.6	51.0	1.5	12807.1	352.3	40.1	1.2	15480.0	423.6	**18.7**	**0.6**
32	14	0.5	16645.7	455.3	3443.9	106.1	1133.7	31.1	12649.1	346.9	1100.2	30.2	14248.9	390.0	**865.0**	**23.7**
33	15	0.05	14055.2	2894.7	478.3	260.5	0.6	0.3	14026.8	2886.9	**0.5**	**0.2**	14044.7	2890.9	0.6	0.3
34	15	0.10	14055.2	2894.7	990.3	376.8	32.3	7.9	14036.3	2889.9	**27.0**	**6.6**	14041.5	2891.2	29.9	7.3
35	16	0.2	23163.9	1630.5	1542.6	192.0	8.9	8.9	22050.6	1581.7	8.0	0.8	23166.6	1631.0	**7.8**	**0.8**
36	16	0.25	23163.9	1630.5	2584.0	279.9	56.7	4.5	22655.7	1607.5	49.8	3.9	23188.1	1632.6	**48.4**	**3.8**
37	17	0.15	30203.5	1943.9	3640.3	354.7	4.1	0.4	29615.3	1924.0	2.6	0.3	30207.2	1944.9	**2.5**	**0.3**
38	17	0.2	30203.5	1943.9	24730.3	1606.6	32.8	2.6	29609.2	1922.7	23.0	1.8	30203.7	1944.6	**22.0**	**1.7**
39	18	0.05	39453.4	2089.1	2669.9	390.5	20.5	1.8	12935.3	1022.9	**1.7**	**0.2**	13195.0	1034.6	9.6	0.9
40	19	0.01	18746.9	2774.9	67.5	36.8	3.4	0.4	16127.1	2667.8	**0.0**	**0.0**	17880.9	2730.0	0.9	0.1
41	20	0.1	14060.1	1214.5	1601.7	36.8	2.0	0.3	10742.4	1007.6	1.7	0.2	14048.0	1213.2	**1.5**	**0.2**
42	20	0.15	14060.1	1214.5	1963.8	291.7	431.9	37.8	15223.8	1309.4	307.2	26.9	15241.1	1311.0	**283.1**	**24.7**