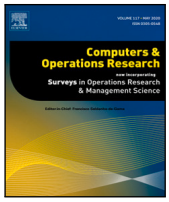
Computers & Operations Research 137 (2022) 105545

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor



Dynamized routing policies for minimizing expected waiting time in a multi-class multi-server system

Vahid Nourbakhsh, John Turner *

The Paul Merage School of Business, University of California, Irvine, CA 92697, United States of America

ARTICLE INFO

Keywords:
Expected queue waiting time
Routing policy
Convex programming
Transportation on demand
Call center

ABSTRACT

Minimizing queue waiting time in multi-class multi-server systems, where the service time depends both on the job type and the server type, has wide applications in transportation systems such as emergency networks and taxi networks, service systems such as call centers, and distributed computing platforms. However, the optimal dynamic policy for this problem is not known and remains a hard open problem. In our approach, we develop a math program to model a static variant of this routing problem and use the solution from this math program to construct several novel dynamic policies. In three categories, namely, (i) policies that do not block jobs, (ii) policies that block jobs statically (i.e., blocking jobs using a predetermined blocking probability), and (ii) policies that block jobs dynamically (i.e., blocking jobs when all feasible servers are busy), we compare the performance of our policies with *Fastest-Server-First* (FSF), a well-known routing policy for such problems in practice and in the literature. Our experiments show that our proposed overflow dynamic routing policies outperform FSF and its extensions, FSFStaticBlock and FSFDynamicBlock. Moreover, to showcase our methodology, we apply our proposed policies to the problem of assigning fire incidents in Irvine, CA, to fire stations.

1. Introduction

We study the problem of minimizing expected waiting time in a multi-class multi-server queueing system, where 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, i.e., the server to which the job is routed. This endogeneity of 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). Nevertheless, simple greedy dynamic policies can perform reasonably well in such settings, and provide a meaningful practical benchmark. In our context, a simple greedy dynamic policy which is commonly implemented in practice is the *Fastest-Server-First* (FSF) policy, which always routes each arriving job to an available (i.e., not busy) server that is the fastest at handling this job type. While the FSF policy 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 will demonstrate that its performance is suboptimal when there are multiple job types.

On the other hand, static policies derived from the solution of math programs can match 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. We dynamize our static policy using FSF as our guide, and using a comprehensive set of simulated instances, we evaluate the performance of our dynamized policies and empirically show that several of our dynamized policies outperform FSF.

The routing problem that we consider 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*). Our problem is quite general, and is at its core a routing problem for a multi-class multi-server system with Poisson arrivals and service rates that are exponentially-distributed and depend on both the job type and server type. We seek to minimize the jobs' *Expected queue Waiting time* (EW).

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

* Corresponding author.
E-mail addresses: vahid.nourbakhsh@uci.edu (V. Nourbakhsh), john.turner@uci.edu (J. Turner).

https://doi.org/10.1016/j.cor.2021.105545
Received 28 April 2018; Received in revised form 30 July 2021; Accepted 30 August 2021
Available online 4 September 2021

(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 gives rise to job-and-server dependent service rates since the service rate depends on the distance between the vehicle base and the requester (cf., 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 each job type. 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 and Boxma (1994). In this context, job-and-server dependent service times can occur for two different reasons, namely (i) users and processors are geographically distributed and consequently latency (the time to transfer a job to a processor) is job-and-server dependent, and (ii) processors are of different speeds and jobs are of different types.

Our main contributions are as follows. Combé and 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 several of our dynamized policies beat the well-known FSF policy and its variants. While the focus of this study is on designing a dynamic policy, our proposed math program can also be extended by incorporating general convex constraints, which allows for the solution of planning problems such as (i) determining the optimal number of servers that guarantee a desired expected waiting time, or (ii) determining the optimal location of server groups. Moreover, in Supplementary Appendix C, we extend our results and show that similar dynamic policies can be easily derived for the performance metrics *Expected Sojourn time* (ES), i.e., the expected time in system defined as queue waiting time plus service time, and *Expected Throughput* (ET), i.e., the expected number of jobs that are not blocked upon arrival and consequently receive service.

The rest of this paper is organized as follows. In Section 2 we review the related literature, in Section 3 we formally define our problem, and in Section 4 we characterize the optimal static policy. In Section 5 we introduce several dynamized routing policies, and in Section 6, we conduct computational experiments to compare these policies. Finally, we showcase the application of dynamized policies for routing fire engines to incident locations in Irvine, California in Section 7. Extensions of our model and conclusions follow in Supplementary Appendix C and Section 8, respectively.

2. Literature review

As mentioned earlier, 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 (cf., Mehrotra et al., 2012). While most of the research on dynamic policies in this context concerns a single job type, we review some of the most related papers that also have service times that depend on the routing decision.

For the special case with one job type and multiple heterogeneous servers (i.e., the 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. Moreover, also in a single-class with multiple heterogeneous servers setting, Armony and 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. As we discuss in Section 4.1, our method is also capable of handling fairness constraints such as limits on workloads or utilizations.

Under a many-server asymptotic regime, Tezcan and 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 type (i.e., not job-and-server dependent).

In a seminal work, Mandelbaum and Stolyar (2004) showed that the generalized $c\mu$-rule (i.e., the $Gc\mu$-rule) asymptotically minimizes convex holding costs. 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 and Tezcan (2008) discuss that applying the $Gc\mu$-rule to linear holding costs can lead to system load explosion.

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 (cf., Mehrotra et al., 2012; Gopalakrishnan et al., 2016). Dai and Tezcan (2008) provide some insight into the challenge of finding the optimal dynamic policy for this problem.

Chan et al. (2014) propose more complex dynamic routing policies than FSF, applied to a variety of sophisticated performance measures such as a convex combination of job waiting time and server idle time. Although the metrics themselves are practical, their use in a routing optimization problem presents challenges as the optimization problem is non-convex. The authors propose a Modified Genetic Algorithm to solve what they call a *black box*-type problem for optimizing (tuning) the parameters of their routing policies. In this paper, our objective is to show the benefits that come from "dynamizing" an optimal solution to a static optimization problem. Consequently, we insist that our routing optimization problem is convex, and make modeling assumptions consistent with this convexity requirement.

Furthermore, as noted in the introduction, since our optimization problem is convex, it can be efficiently embedded into larger planning problems that also tackle staffing and scheduling concerns. The convexity of our model improves the solution efficiency of such problems. With regard to call center staffing, the interested reader may wish to consult Ta et al. (2016) which covers a number of important issues in that problem.

Finally, although our study is limited to queuing systems with no abandonment, in a sense blocking (which we model, both static and dynamic variants) has a similar effect on keeping the workload of the system stable. For a more explicit treatment of dynamic abandonment, see Pichitlamken et al. (2003) who model a system in which a caller's patience time follows an exponential distribution with a mean that changes during the day.

3. Problem

In our multi-class multi-server 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 stays 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 $j \in J_i$, where J_i is the set of server groups that are eligible to serve jobs of type i. At each server group j, there are k_j identical servers. Service times are independent, each exponentially distributed with mean service time τ_{ij}.

For completeness, we will also denote J as the full set of server groups (henceforth known simply as groups), I_j as the set of job types that can be served by group j, and $F = \cup_{i\in I, j\in J_i}\{(i,j)\}$ as the

set of tuples (i, j) representing all feasible job-type to server-group assignments. Note also that in the call center literature, the terms job type and server group are referred to as call type and agent group, respectively.

In this setting, we seek to minimize the expected 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. This decision may consider the current availability of the servers, as well as the fact that the expected service time τ_{ij} depends on both the job type i and the server group j.

In the policies that follow, the queues where jobs wait for service can either be before or after jobs are routed to a specific server group. Consequently, the placement of queues within the model is a policy-specific decision, and is omitted from the formal problem definition.

4. 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 4.1 we formulate a math program which we use to compute the aggregate volume of jobs that should be assigned to each server group. Then, in Section 4.2 we introduce XRand, a static routing policy constructed from the solution of our math program.

4.1. 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 the service discipline is First-Come First-Served (FCFS). Moreover, as described earlier in Section 3, 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, via the x_{ij} variables, to control the workload sent to each queue.

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 is computed using the Erlang-C function (Cooper, 1981) defined as,

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

where the symbol (!) is the factorial function, r_j denotes the workload of group j as defined in (3) below, and k_j is the number of servers in group j irrespective of their status, i.e., busy or available. Although this formula is exact only when we assume exponential service times, we expect our model to be useful even when service times are not exponential since others (cf., Kimura, 2010) have shown the Erlang-C function tends to be a good approximation for the delay probability 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,

$$\lambda_j = \sum_{i \in I_j} x_{ij} \quad \forall j \in J \quad . \tag{2}$$

The workload of group j is,

$$r_j = \sum_{i \in I_j} \tau_{ij} x_{ij} \quad \forall j \in J \quad . \tag{3}$$

The mean service time at group j is,

$$\sigma_j := \frac{r_j}{\lambda_j} \quad \forall j \in J \quad . \tag{4}$$

Table 1
Model notation.

Indices and sets	
$i \in I$	Index for job types; set of all job types
$j \in J$	Index for server groups; set of all server groups
I_j	The subset of job types that group j can serve
J_i	The subset of groups that can serve jobs of type i
F	Set of tuples (i, j) representing all feasible job-type to server-group assignments
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^{param}	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
σ_j	Mean service time at group j

The model is schematically depicted in Fig. 1 with the notation summarized in Table 1.

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 the M/M/k queue at group j, EW can be written as the delay probability multiplied by the expected delay conditional on the job being delayed (cf., Hokstad, 1978),

$$EW(\lambda_j, r_j) = EC(k_j, r_j) \frac{r_j}{\lambda_j (k_j - r_j)}, \tag{5}$$

where the Erlang-C function $EC(k_j, r_j)$ is defined in (1) and provides the delay probability. While the above function measures the expected waiting time of jobs at a single group (i.e., at the queue before a 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,

$$EW_{avg} := \frac{\sum_{j \in J} \lambda_j EW(\lambda_j, r_j)}{\sum_{j \in J} \lambda_j}, \tag{6}$$

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

$$EW_{tot} := \sum_{j \in J} \lambda_j EW(\lambda_j, r_j). \tag{7}$$

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,

$$\begin{aligned}
(P1) \quad & \min \quad EW_{avg} \\
& \text{s.t.} \quad \text{Constraints (2) and (3)}, \\
& \qquad r_j \le k_j + \epsilon \quad \forall j \in J, && (8) \\
& \qquad \sum_{j \in J_i} x_{ij} \le d_i \quad \forall i \in I, && (9) \\
& \qquad \text{Optional convex constraints}, && (10) \\
& \qquad x_{ij} \ge 0 \quad \forall (i, j) \in F. && (11)
\end{aligned}$$

Constraint (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. Since strict inequalities cannot be present in math programs, we introduced a new parameter, $\epsilon > 0$, and write this inequality as $r_j \le k_j + \epsilon$ instead of $r_j < k_j$. Any reasonable value for ϵ, such as $\epsilon = 0.001$, will prevent the queues from becoming unstable. Constraint (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

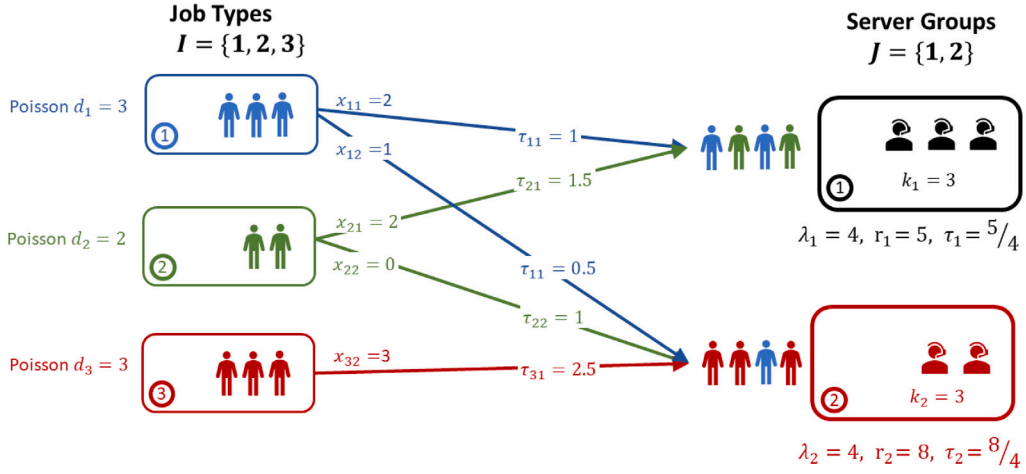


Fig. 1. Schematic representation of the multi-class multi-server routing system modeled by our math program.

i. Because Constraint (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 (10) indicates that one has the option to impose additional constraints that are convex in the variables x_{ij}, r_j, and λ_j. For example, we can make sure that the workload is fairly distributed between groups by imposing lower and upper bounds on the r_j's. Or, we can embed $(P1)$ into a larger math program to link additional decisions. Finally, Constraint (11) ensures all x_{ij} variables are 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 be degenerate, have no congestion, and not serve any job. Thus, we add Constraint (12), which exogenously defines a global coverage level,

$$\sum_{j\in J} \lambda_j \geq CF^{param} \sum_{i\in I} d_i, \tag{12}$$

where $0 \leq CF^{param} \leq 1$ is a parameter we call the global coverage factor.

Given a feasible instance of $(P1)$ with Constraint (12) in place, then Constraint (12) binds at optimality (See Proposition 1 below). Indeed, one cannot improve EW_{avg} by increasing the coverage, $\sum_{j\in J} \lambda_j$, beyond the minimum required coverage, $CF^{param} \sum_{i\in I} d_i$.

Proposition 1. *Assuming that problem $(P1)$ with Constraint (12) is feasible, the coverage Constraint (12) is binding.*

Proof. See Appendix A. □

Proposition 1 indicates that at optimality EW_{avg} becomes,

$$EW_{avg} = \frac{\sum_{j\in J} \lambda_j EW(\lambda_j, r_j)}{\sum_{j\in J} \lambda_j} = \frac{\sum_{j\in J} \lambda_j EW(\lambda_j, r_j)}{CF^{param} \sum_{i\in I} d_i} = \frac{EW_{tot}}{CF^{param} \sum_{i\in I} d_i}. \tag{13}$$

Since $CF^{param} \sum_{i\in I} d_i$ is a constant, optimizing EW_{avg} in Problem $(P1)$ with Constraint (12) is equivalent to optimizing EW_{tot} with Constraint (12). Consequently, although we are most interested in solving $(P1)$ with the EW_{avg} objective and Constraint (12), we will do this by solving $(P2)$ defined as,

$$\begin{aligned} (P2) \quad & \min \quad EW_{tot} \\ & \text{s.t.} \quad \text{Constraints (2), (3), (8), (9), (10) and (11),} \\ & \qquad \sum_{j\in J} \lambda_j = CF^{param} \sum_{i\in I} d_i. \end{aligned} \tag{14}$$

The following proposition proves that EW_{tot} is non-linear but convex in r_j, which makes it amenable to numerical optimization. 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. *$EW_{tot} = \sum_{j\in J} \lambda_j EW(\lambda_j, r_j)$ is convex in r_j and independent of λ_j.*

Proof. See Appendix B. □

In particular, for the 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,

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

Notice that although EW_{tot} is nonlinear and non-quadratic, it is can be written as a convex function of the r_j variables, which makes it amenable to convex optimization.

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

4.2. XRand : The optimal static policy

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

Note that Mehrotra et al. (2012) implemented a similar policy they call XRand, where they obtain routing probabilities from solving a slightly different mathematical program. While we minimize EW_{tot} in Problem $(P2)$, Mehrotra et al. (2012) maximize total call resolution rate.

XRand is a static policy that does not consider the state of the system (i.e., the availability of servers) when making routing decisions. Its usefulness stems from the interpretation of x_{ij}'s as scaled routing probabilities. On the other hand, dynamic policies that use the actual real-time state of the system to choose how to route jobs to servers would outperform XRand. In the following section, we introduce routing policies that use *both* the x_{ij}'s computed from solving $(P2)$ *and* the

actual real-time state of the system to dynamically assign jobs to server groups. As we will see, the x_{ij}'s contain important information about how to assign workloads to server groups in aggregate that is useful for constructing a well-performing dynamic policy.

5. 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). Then, in the following section we will empirically compare the performance of all policies.

Recall that to construct the math program ($P2$) in Section 4.1, we assumed that queueing occurred after the routing decision, i.e., each server had a queue in front of it. 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 Fig. 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 the resulting dynamic policies.

We divide the dynamic routing policies into three categories as follows: (i) policies that do not block jobs, (ii) policies that block jobs statically, and (iii) policies that block jobs dynamically.

5.1. Dynamic routing policies with no blocking

The first three dynamic policies that we describe do not block any jobs. While serving all jobs that arrive may be an important criterion for some systems, expected waiting time can suffer when the inability to block jobs leaves the system congested.

5.1.1. FSF policy

We begin describing the Fastest-Server-First (FSF) policy, since it is both a benchmark and its logic is incorporated into some of our policies. This intuitive policy is known as an *overflow policy*, as its operation can be described by job-to-group and group-to-job priority lists as follows. 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 (if a free server is found, the job is assigned to it). This priority list is an ordered list of all groups that can serve job i, i.e., $j \in J_i$, sorted from smallest to largest service times τ_{ij}. The policy is called FSF because the fastest server group has the highest priority in the list, and therefore if the fastest server is free it will always be the one assigned. There is one queue for each job type $i \in I$, and if a job of type i finds all groups in J_i busy, it will enter queue i and be served in FCFS order relative to other jobs of type i. Moreover, when a server in group j becomes free and there are waiting jobs in at least one queue $i \in I_j$, a group-to-job list for that server group determines which waiting job is picked. Specifically, the group-to-job list for group j is a list of all job types that server group j can serve, i.e., $i \in I_j$, again sorted from smallest to largest service times τ_{ij} (i.e., from the server's perspective, the policy follows a "fastest job-type first" rule by picking the waiting job that is the fastest to process). If all queues in the list are empty, then the server stays free. Finally, without loss of generality, if fairness to individual servers at a group is a consideration, then whenever more than one server is idle at the assigned group, we may choose the server with the longest idle time, i.e., follow the Longest Idle Server First (LISF) rule.

5.1.2. XOverflow policy

We construct an overflow policy that, like FSF, uses priority lists to determine routings. However, instead of determining priorities by rank-orders of service time τ_{ij}, our XOverflow policy 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}. We omit matching groups and job types whose x_{ij} values are zero, since the solution to ($P2$) suggests no jobs of type i should be processed by group j. Consequently, the job-to-group list for job type i may have fewer members than $|J_i|$, and the group-to-job list for group j may have fewer members than $|I_j|$.

5.1.3. XFSF policy

This overflow policy is a combination of the XOverflow and FSF policies. First, we initialize the priority lists so that they are the same as in the XOverflow 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. In essence, the priority lists use x_{ij} as a primary criterion to rank-order matching groups and jobs whenever x_{ij} is strictly positive, and τ_{ij} as a secondary criterion to rank-order matching groups and jobs for those groups and jobs that have $x_{ij} = 0$. As a consequence, the priority lists may be longer than those used by the XOverflow policy. Indeed, the job-to-group list for job type i always has $|J_i|$ members, and the group-to-job list for group j always has $|I_j|$ members.

5.2. Dynamic routing policies with static blocking

Our next three policies block some jobs from obtaining service, but do so in a purely random fashion without adapting the blocking rate to the current state of the system (i.e., real-time server availability). Compared to not blocking at all, static blocking improves expected waiting time by reducing workloads at server groups and keeping congestion in check. On the other hand, throughput (the number of served jobs) decreases. Whereas our non-blocking policies all had a coverage factor of 1 (i.e., all jobs get served), our static-blocking policies block jobs so that in expectation, the actual coverage factor matches the desired coverage factor, CF^{param}. Note that the optimal static routing policy, XRand, also blocks jobs in a static fashion, and so it is interesting to compare the performance of the policies in this section to that of XRand.

5.2.1. XOverflowStaticBlock policy

Upon arrival of a job of type i, with probability $\bar{C}_i/d_i$ we block the job and do not serve it. This static blocking probability is the same as defined by XRand, and requires the solution to the math program ($P2$) to compute the planned number of blocked jobs for each job type, $\bar{C}_i = d_i - \sum_{j \in J_i} x_{ij}$. Non-blocked jobs are routed according to the priority lists defined by our XOverflow policy.

5.2.2. XFSFStaticBlock policy

This is a variant of our XFSF policy in which arrivals of type i are blocked with probability $\bar{C}_i/d_i$. Non-blocked jobs are routed according to the priority lists defined by our XFSF policy. Note that again, we must solve ($P2$) to compute the planned number of blocked jobs for each job type, $\bar{C}_i$.

5.2.3. FSFStaticBlock policy

This is a variant of the FSF policy which blocks all arriving jobs (irrespective of type) with probability CF^{param}, and routes non-blocked jobs according to the priority lists defined by the regular FSF policy. We use this policy as a benchmark for our static-blocking policies. Like FSF, this policy does not use any information from the solution of ($P2$).

5.3. Dynamic routing policies with dynamic blocking

Our final three policies dynamically block some jobs from obtaining service by using the real-time state of the system (i.e., server availability) to decide when to block a job. While there are potentially many ways to dynamically block jobs, our intent here is to demonstrate that, for a given type of dynamic-blocking policy, we can improve its performance by using information from the solution of the math program ($P2$). Consequently, we establish this result with the most canonical dynamic-blocking policy.[1] Specifically, we set the queue length for each job type to zero, and block type-i jobs whenever no server is available at any server group matching job type i. Since for our dynamic-blocking policies the queue length is zero, expected waiting time is also zero. Therefore, we use the actual coverage factor (percentage of jobs served) to measure the performance of these policies.

5.3.1. FSFDynamicBlock policy

When a job arrives, we traverse the corresponding job-to-group priority list and assign the job to the first server group that has an idle server. If all matching server groups are completely busy, the job is blocked. This is our dynamic-blocking benchmark, as the job-to-group priority lists are ordered from smallest to largest mean service time τ_{ij} just like in the FSF policy, and no information from the math program ($P2$) is used to direct the policy.

5.3.2. XDynamicBlock policy

This is a variant of our XOverflow policy in which arrivals are blocked dynamically. When a job arrives, we traverse the corresponding job-to-group priority list and assign the job to the first server group that has an idle server. If all matching server groups are completely busy, the job is blocked. Priority lists are defined as in our XOverflow policy, i.e., they are ordered from largest to smallest planned allocation volume x_{ij}, where the allocation volumes are computed by solving math program ($P2$).

5.3.3. XFSFDynamicBlock policy

This is a variant of our XFSF policy in which arrivals are blocked dynamically. When a job arrives, we traverse the corresponding job-to-group priority list and assign the job to the first server group that has an idle server. If all matching server groups are completely busy, the job is blocked. Priority lists are defined as in our XFSF policy, i.e., we initialize the priority lists so that groups and jobs are rank-ordered from highest to lowest x_{ij}. Then, we append additional groups (and jobs) to the ends of job-to-group (and group-to-job) lists in order of smallest to largest mean service time τ_{ij}. Note that again, we must first solve ($P2$) to compute the planned allocation volumes x_{ij}.

6. Experiments

In this section, we measure the performance of our routing policies using a simulation. We implemented the convex math program ($P2$) in AMPL 2017.01.26 on a Windows PC with 16 GB of RAM and a quad core CPU running at 3.1 GHz, and solved it using KNITRO 10.1, a commercial solver for convex programs. In Section 6.1 we explain how we generated our test instances, in Section 6.2 we describe the simulation we use to measure the performance of our policies, and finally in Section 6.3 we report our simulation's results.

[1] More advanced dynamic-blocking policies could be constructed by either (a) setting a finite non-zero queue length and blocking when all matching queues are full, or (b) dynamically adapting queue length and/or amount of work admitted to each queue, based on either the global coverage factor CF^{param} or the job type-specific blocking probabilities $\bar{C}_i/d_i$ computed by the math program ($P2$). We leave the analysis and measurement of these more advanced dynamic-blocking policies for future work.

6.1. 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 each instance we generate parameters k_j, d_i and τ_{ij} as well as the adjacency sets I_j and J_i. For summary statistics of all instances see Supplementary Appendix A. Below, we explain how we generate our instances.

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 our non-planar instances, we generate expected service times τ_{ij}'s randomly from a uniform distribution with support on the interval $[0.5, 3.5]$. To generate links (i, j) in the bipartite graph between job types and servers, we proceed as follows. For each (i, j) combination, we generate a uniform random number between 0 and 1 and add the link (i, j) to the graph if this random number is greater than a threshold we call graph density. Graph density is a number between 0 and 1, and is defined as the proportion of links that our graph should have compared to the complete bipartite graph on the same set of nodes (See Table 1 in Supplementary Appendix A for the graph densities of our instances). The adjacency sets J_i and I_j are computed directly from the graph.

For our planar instances, we generate random points for job nodes and group nodes uniformly on a plane measuring 100×100 units. 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., $i \in I_j$, the Euclidean distance between i and j gives us the service time, τ_{ij}.

For many of our instances we consider several coverage factors (CF^{param}). In total, we solve problem ($P2$) and simulate our policies for 42 test cases (i.e., 42 {instance, coverage factor} combinations).

6.2. Simulation design

We implemented our simulation using the *ContactCenters* Java library of Buist and L'Ecuyer (2005). In our simulation, jobs of each type i arrive independently according to Poisson processes with arrival rates 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 report average performance for both expected waiting time, EW_{avg}, and actual coverage factor, i.e., proportion of jobs served as measured by the simulation, CF^{act}.

We use this simulation to compare the performance of our ten policies (the static policy XRand, as well as the nine dynamic policies defined in Section 5). Specifically, we subdivide our ten policies into three categories based on how jobs are blocked as follows:

(i) No-Blocking Policies: FSF, XOverflow, and XFSF. Because these policies serve all jobs, the actual coverage factor CF^{act} is always 1. Performance is measured by expected waiting time, EW_{avg}.

(ii) Static-Blocking Policies: XRand, FSFStaticBlock, XOverflowStaticBlock, and XFSFStaticBlock. Because these policies block a pre-computed fraction of arriving jobs, in expectation (i.e., in the long run) the actual coverage factor will equal the desired coverage factor, i.e., $CF^{act} = CF^{param}$. Performance is measured by expected waiting time, EW_{avg}.

(iii) Dynamic-Blocking Policies: FSFDynamicBlock, XDynamicBlock, and XFSFDynamicBlock. Because for these policies queue length is zero, expected waiting time EW_{avg} is also zero. Performance is measured by actual coverage factor CF^{act}.

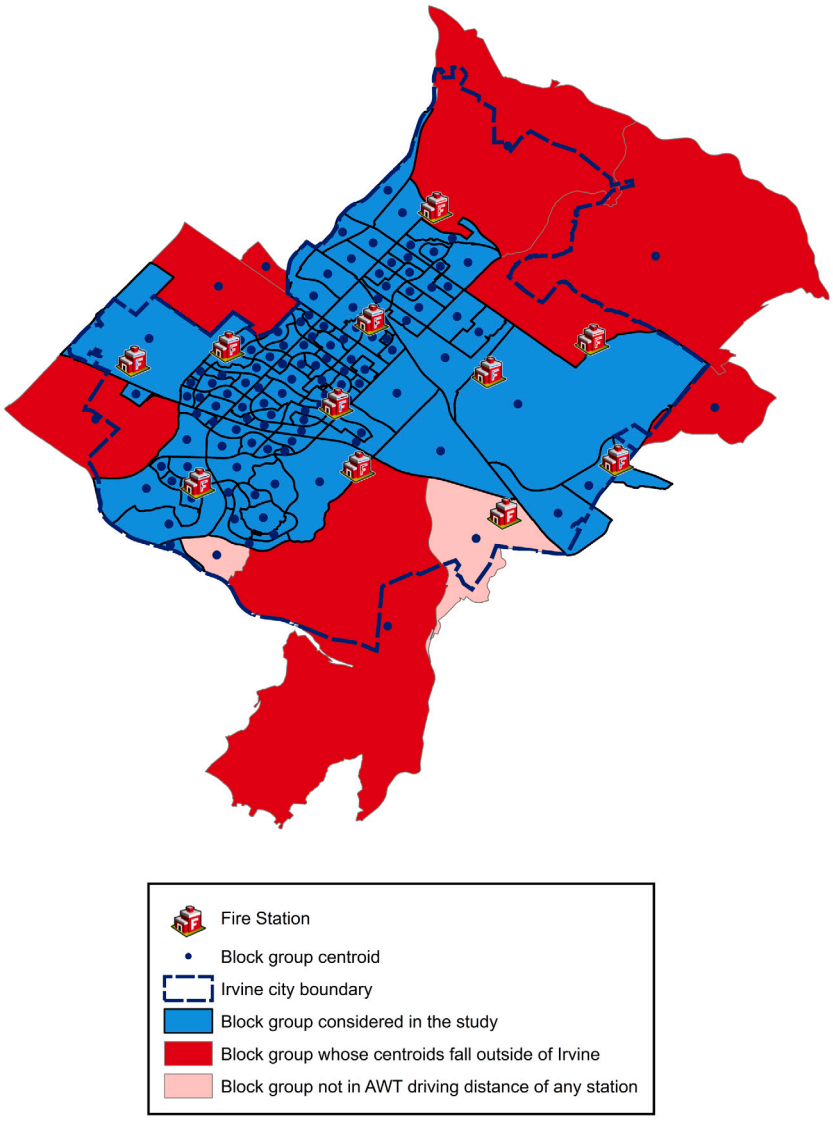


Fig. 2. Irvine fire stations and the census block groups considered in the study. Excluded block groups are those with centroids that are either (i) outside of the city boundary, or (ii) too far from any fire station, i.e., the Acceptable Waiting Time (AWT) of 8 min 45 s would always be exceeded.

6.3. Results

We compare policies in each blocking category separately. For each policy, we report the number of instances (out of 42) that the policy outperforms the other policies in its category. Table 2 summarizes our results. Note that because two or more policies can be best-performing for a given instance, within a category the sum of percentages in the right-most column may exceed 100%. Numerical results for all 42 test cases in the non-blocking, static-blocking, and dynamic-blocking tests are provided in the Supplementary Appendix B (see Tables 2, 3, and 4, respectively).

Among policies that do not block jobs (i.e., FSF, XOverflow, and XFSF), XOverflow outperforms the other two policies. In the static-blocking category, XOverflowStaticBlock and XFSFStaticBlock are the best-performing polices. Finally, XFSFDynamicBlock outperforms among the dynamic-blocking policies.

In all 42 test cases, FSF and its variants (FSFStaticBlock and FSFDynamicBlock) are dominated by one or more of our dynamized policies. We attribute this result to the fact that FSF and its variants are myopic and decentralized, i.e., each job chooses the fastest available server without considering other jobs, while our dynamized policies which 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, but keeps the fastest available server for a future incoming job or even a job of a different type. Indeed, our dynamized policies try to balance server workloads to benefit the whole system, and are less greedy.

Table 2
Summary of results: Number of instances (out of 42 instances) that each policy is best-performing in its category.

Policy category	Performance criterion	Policy	Number of instances that the policy is the best.
No blocking ($CF^{act} = 1$)	EW_{avg}	FSF	4 (9.5%)
		XOverflow	34 (81.0%)
		XFSF	7 (16.7%)
Static blocking ($CF^{act} = CF^{param}$)	EW_{avg}	XRand	4 (9.5%)
		FSFStaticBlock	2 (4.8%)
		XOverflowStaticBlock	25 (59.5%)
		XFSFStaticBlock	27 (64.3%)
Dynamic blocking ($EW_{avg} = 0$)	CF^{act}	FSFDynamicBlock	4 (9.5%)
		XDynamicBlock	16 (38.1%)
		XFSFDynamicBlock	38 (90.5%)

Our simulation results show that the solution to our math program ($P2$) is useful for building dynamized routing policies for online settings. Particularly, XOverflow, XFSFStaticBlock, XOverflowStaticBlock, and XFSFDynamicBlock are all well-performing policies in their categories. This illustrates that there is a 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 such as FSF 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.

Finally, we also point out that blocking some jobs upon arrival can be important to keep servers from becoming overloaded. In general, finding the optimal blocking level for each job type is non-trivial because service rates are job-server dependent and servers' workloads depend on the routing. Another use of our math program ($P2$) is to compute job-type specific coverage ratios, which can be interpreted as job-type specific blocking probabilities, $\bar{C}_i/d_i$. These were used by the static policy XRand, as well as the dynamic-routing-with-static-blocking policies XOverflowStaticBlock and XFSFStaticBlock.

7. Fire stations case

To illustrate the applicability of our methodology, we now assign incidents to fire stations in Irvine, California. We describe the data in Section 7.1, visually depict some key routing policies in Appendix C, and finally compare the numerical performance of routing policies in Section 7.2.

7.1. Data

For emergency calls, the Orange County Fire Authority (OCFA) dispatches vehicles to incident locations in Orange County, California. 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 Authority, 2014). In 2010, OCFA received 85,212 emergency calls; we allocate these to 1822 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 each incident occurs at the centroid of the census block group where it originates; see Fig. 2 for a map of Irvine's census block groups and their centroids. We make this assumption primarily for tractability, as the problem would be too large if we modeled the exact location of each incident. In any case, the exact locations of past incidents may not represent the locations of future incidents, and so one could view using the centroids as a reasonably robust alternative. Among the 1822 block groups in OC, 124 block groups either completely fall in or intersect with the boundaries of the city of Irvine. Among these 124 block groups, 117 have centroids that fall in Irvine (See blue and orange block groups in Fig. 2), and 7 have centroids that do not fall in Irvine (See red block groups in Fig. 2). 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 XRand policy with the dynamic policies we developed in Section 5 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. We refer the reader to L'Ecuyer et al. (2018) for a more explicit treatment of bursty arrivals coming from correlated emergencies that spawn from a single event such as a natural disaster.

Table 3
Simulation results for the Irvine fire stations case.

Policy type	Routing policy	Performance measure	CF^{param}[a]		
			0.80	0.85	0.90
No blocking	FSF[c]	CF^{act}[b]		1	
		EW_{avg}		0.51	
	XOverflow	CF^{act}		1	
		EW_{avg}	0.44	0.49	0.75
	XFSF	CF^{act}		1	
		EW_{avg}	**0.43**[d]	**0.46**	**0.56**
Static	FSFStaticBlock	CF^{act}	0.80	0.85	0.90
		EW_{avg}	0.49	0.50	0.50
	XRand	CF^{act}	0.80	0.85	0.90
		EW_{avg}	0.45	0.51	0.74
	XOverflowStaticBlock	CF^{act}	0.80	0.85	0.90
		EW_{avg}	0.08	0.13	0.17
	XFSFStaticBlock	CF^{act}	0.80	0.85	0.90
		EW_{avg}	**0.08**[d]	**0.12**	**0.16**
Dynamic	FSFDynamicBlock	CF^{act}		0.68	
		EW_{avg}		0[e]	
	XDynamicBlock	CF^{act}	0.60	0.60	0.60
		EW_{avg}	0[e]	0	0
	XFSFDynamicBlock	CF^{act}	**0.74**[d]	**0.77**	**0.77**
		EW_{avg}	0[e]	0	0

[a] CF^{param} is the coverage factor parameter used in the math program ($P2$).
[b] CF^{act} is the actual coverage factor measured by the simulation.
[c] The coverage factor parameter CF^{param} does not apply to the FSF policy.
[d] **Bold face** denotes the best routing policy in each blocking type category.
[e] Recall queueing does not occur in our dynamic-blocking policies (jobs are blocked when all feasible groups are busy).

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 min 45 s (Orange County Fire Authority, 2014). Using this acceptable waiting time, we build sets J_i and I_j, i.e., block group i is covered by station j if centroid i is within 8 min 45 s driving distance of j. There are two centroids that are not within the 8 min 45 s driving distance of any fire station and thus are not considered in our experiment (See light red block groups in Fig. 2).

For visualization, we illustrate policies XRand and XFSFStaticBlock in Appendix C figures.

7.2. Results

We simulate each of the dynamic policies mentioned in Section 5 as well as our static policy XRand, and list the results in Table 3. Among policies that do not block jobs, we find XFSF is the best routing policy, followed closely by XOverflow. Among the static-blocking policies, XFSFStaticBlock is the best routing policy, followed closely by XOverflowStaticBlock. Finally, XFSFDynamicBlock is the best dynamic-blocking policy. These results are generally consistent with our simulation results from Section 6. Of note, we see that although XRand maps workloads to groups and chooses blocking rates using the solution of ($P2$), it underperforms several other policies. Yet, by judiciously using information coming from the solution of ($P2$), our dynamized policies can outperform both the static policy XRand and simple dynamic policies based on FSF.

Finally we illustrate XFSFStaticBlock in Fig. C.4, which shows the first three priorities for each block group. Other policies can be mapped similarly.

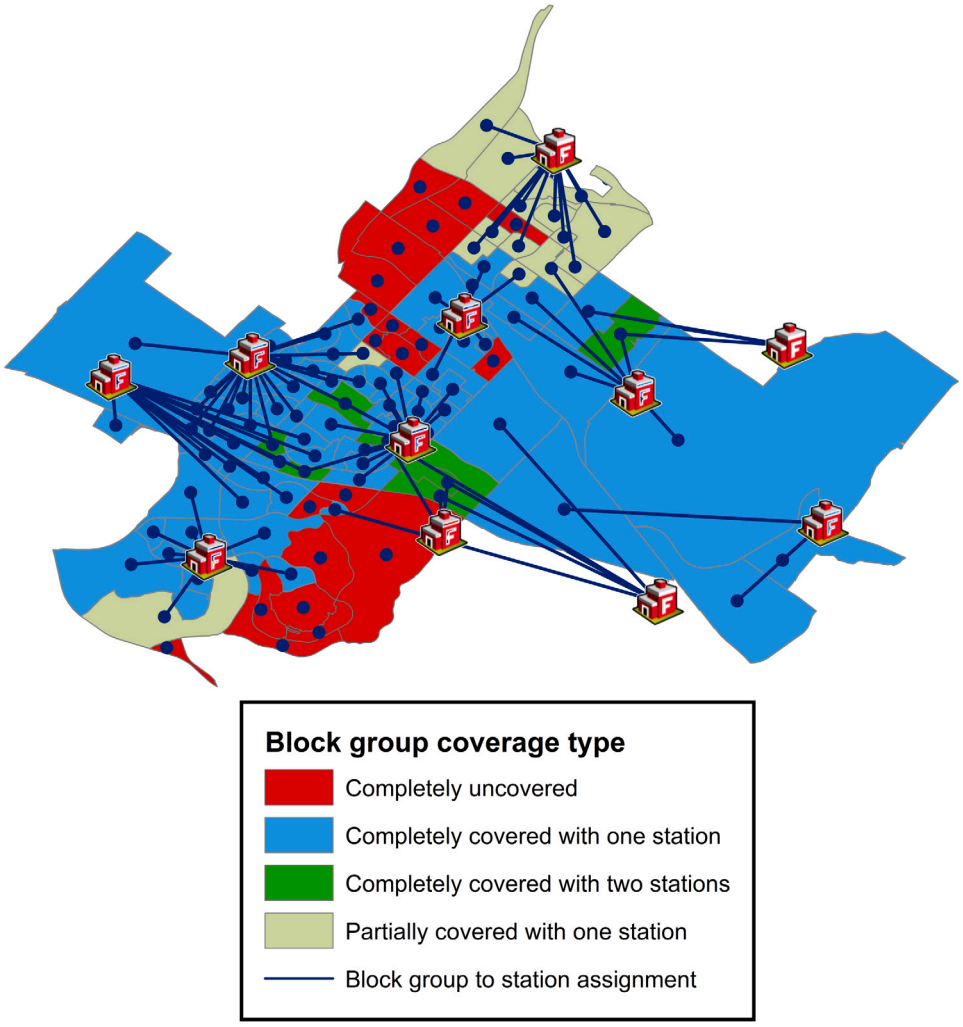


Fig. C.3. Block groups assigned to stations based on the XRand policy. The XRand policy does not cover all the block groups. Some blocks groups are partially covered and some are completely uncovered. Also, there are block groups that are covered by more than one station. In these cases, the XRand policy randomly picks a station for each incident according to the solution of problem ($P2$). Alternatively, one can divide such block groups into zones and assign each zone to one station. This is depicted in Fig. D.5 in Appendix D.

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 well-performing dynamic policies that outperform Fastest-Server-First (FSF) and its extensions, namely, FSFStaticBlock and FSFDynamicBlock.

In Supplementary Appendix C we show 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 outperform the FSF policy, which is a common routing policy used in the literature and in practice for multi-class multi-server systems. Moreover, the dynamic policies we construct are overflow policies that are easy to implement. 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.

CRediT authorship contribution statement

Vahid Nourbakhsh: Methodology, Software, Data curation, Formal analysis, Writing - original draft. **John Turner:** Conceptualization, Methodology, Writing - review & editing, Supervision.

Appendix A. Proof of Proposition 1 : Binding coverage constraint

Proof. 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 (12) is not binding, i.e.,

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

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

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

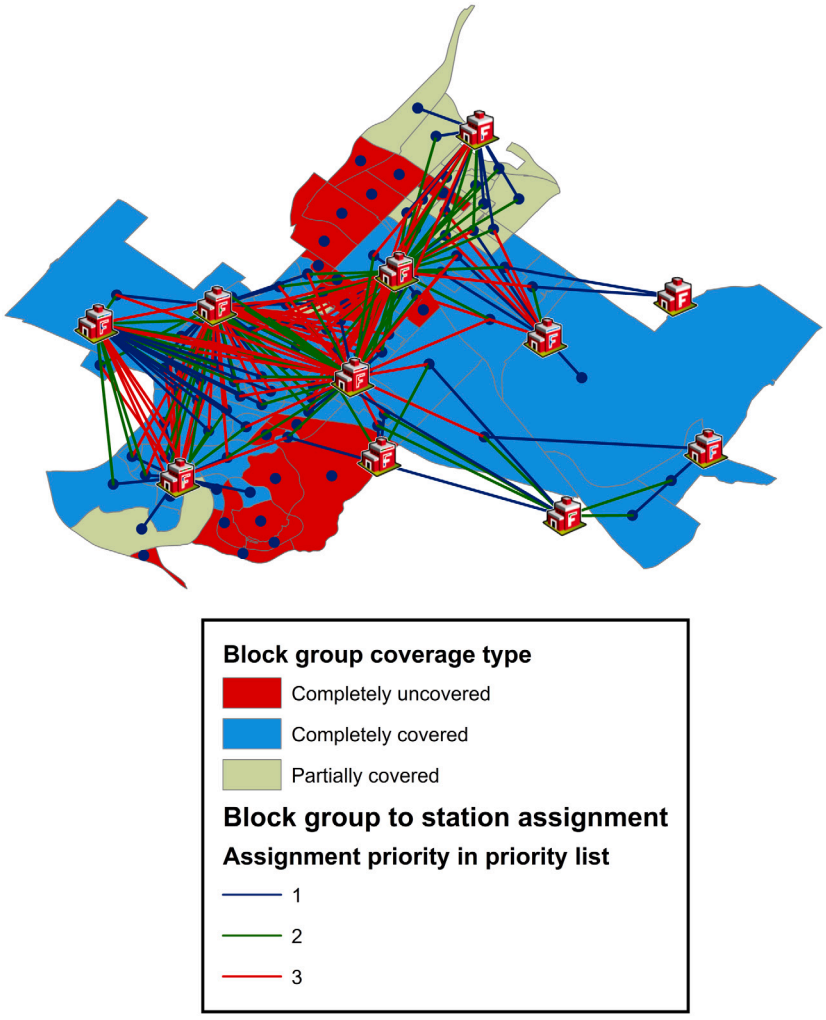


Fig. C.4. Block groups assigned to stations based on the XFSFStaticBlock policy. Similar to the FSF policy depicted in Fig. C.3 the router checks blue, green, and red links in this order for an available fire engine.

where,

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

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

$$\bar{\sigma}_j = \frac{\bar{r}_j}{\bar{\lambda}_j} = \frac{\beta r_j^*}{\beta \lambda_j^*} = \sigma_j^* \quad \forall j \in J.$$

Lemma 3 (which follows) proves that $EW(\lambda, r)$ is strictly increasing in λ and r with fixed mean service time $\sigma > 0$. Thus,

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

Using inequalities (A.1) and (A.2), the following inequality follows,

$$\overline{EW}_{avg} = \frac{\sum_{j\in J} \bar{\lambda}_j EW(\bar{\lambda}_j, \bar{r}_j)}{\sum_{j\in J} \bar{\lambda}_j} = \frac{\sum_{j\in J} \lambda_j^* EW(\bar{\lambda}_j, \bar{r}_j)}{\sum_{j\in J} \lambda_j^*}$$

$$< \frac{\sum_{j\in J} \lambda_j^* EW(\lambda_j^*, r_j^*)}{\sum_{j\in J} \lambda_j^*} = EW_{avg}^*,$$

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

Lemma 3. *Expected waiting time $EW(\lambda, r)$ is strictly increasing in λ and r with fixed mean service time $\sigma > 0$.*

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

Appendix B. Proof of Proposition 2: EW_{tot} Convexity

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

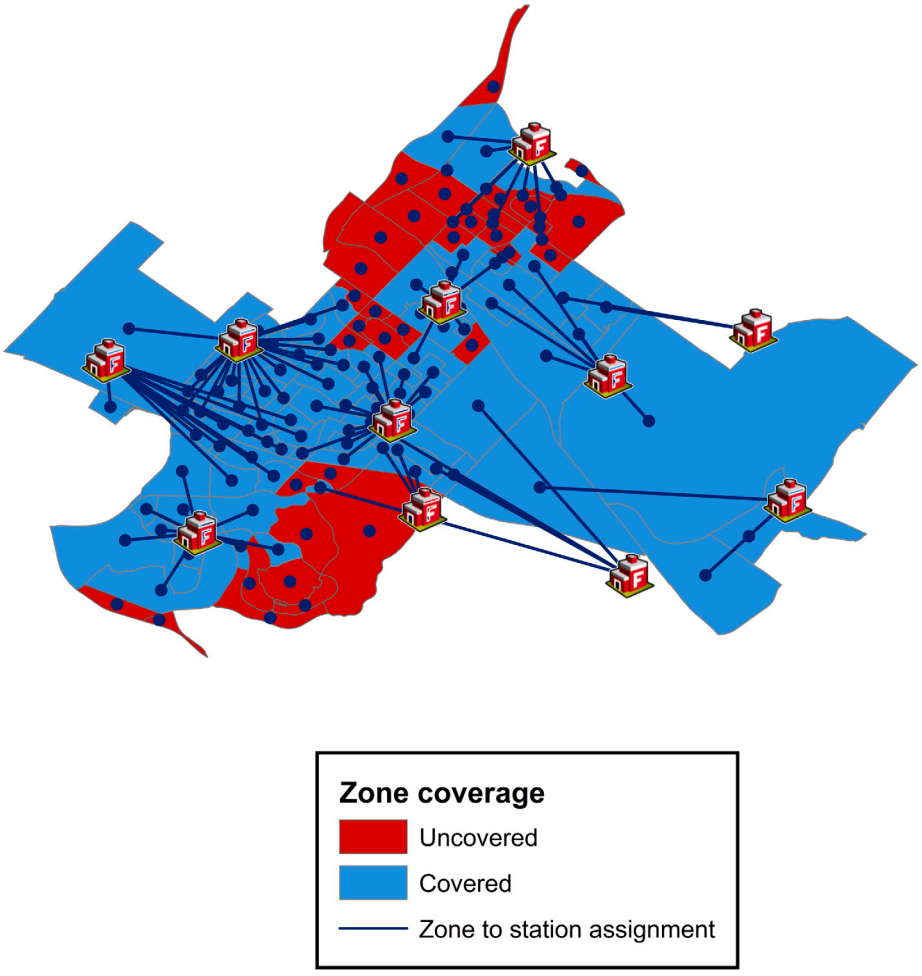


Fig. D.5. Static routing map corresponding to the XRand map in Fig. C.3.

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 σ_j and λ_j do not appear in $E[L_j]$. □

Appendix C. Illustrations of key routing policies

We depict the XRand policy in Fig. C.3. Recall that for the XRand policy, when an incident in block group i occurs it is randomly routed to station $j \in J_i$ with a probability proportional to x_{ij}. In practice, such random assignments may be undesirable. However, if desired we can build an equivalent map that is not subject to randomization but instead subdivides each block group into a number of smaller zones. We produce one zone for each station j with non-zero x_{ij} value, and assign all incidents within that zone to station j. Moreover, we choose the boundary of each zone such that the proportion of the block group's incidents occurring in the zone is equal to the proportion of incidents that would be assigned to station j in the equivalent randomized policy. Note that we can also use this scheme to subdivide a region subject to static (probabilistic) blocking into two subregions: one where no arrivals are blocked and another where all arrivals are blocked. For further details, see Appendix D. Finally, it is worth mentioning that blocking in practice could mean that another fire management system outside our jurisdiction takes over, or that a dispatcher takes all calls and rejects a call if the call is deemed redundant, irrelevant or not urgent. Our exposition is agnostic to these details.

Appendix D. XRand map and the corresponding static routing map

In the XRand policy, when an incident in block group i occurs it is randomly routed to $j \in J_i$ 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 XRand. 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 do not 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 XRand 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 XRand policy (See Fig. C.3). As depicted in Fig. C.3 block group i is linked to station $j \in J_i$, if x_{ij} is strictly positive. There are multiple ways to split a block to multiple zones and Fig. D.5 shows one static routing map corresponding to the map in Fig. C.3. In Fig. D.5 each zone is linked to at most one station.

Online Supplementary Appendix. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cor.2021.105545. This includes (a) detailed descriptions of simulation instances, (b) detailed simulation results, (c) model extensions to optimize Expected System time (waiting time plus service time) and Expected Throughput (flow rate of non-blocked jobs).

References

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

Armony, M., Ward, A.R., 2010. Fair dynamic routing in large-scale heterogeneous-server systems. Oper. Res. 58 (3), 624–637.

Buist, E., L'Ecuyer, P., 2005. A java library for simulating contact centers. In: Proceedings of the 37th Conference on Winter Simulation. Winter Simulation Conference, pp. 556–565.

Chan, W., Koole, G., L'Ecuyer, P., 2014. Dynamic call center routing policies using call waiting and agent idle times. Manuf. Serv. Oper. Manage. 16 (4), 544–560.

Cho, S.-H., Jang, H., Lee, T., Turner, J., 2014. Simultaneous location of trauma centers and helicopters for emergency medical service planning. Oper. Res. 62 (4), 751–771.

Combé, M., Boxma, O.J., 1994. Optimization of static traffic allocation policies. Theoret. Comput. Sci. 125 (1), 17–43.

Cooper, R.B., 1981. Introduction to Queueing Theory. North Holland.

Cordeau, J.-F., Laporte, G., Potvin, J.-Y., Savelsbergh, M.W., 2007. Chapter 7 transportation on demand. In: Barnhart, C., Laporte, G. (Eds.), Transportation. In: Handbooks in Operations Research and Management Science, vol. 14, Elsevier, pp. 429–466.

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

Gans, N., Koole, G., Mandelbaum, A., 2003. Telephone call centers: Tutorial, review, and research prospects. Manuf. Serv. Oper. Manage. 5 (2), 79–141.

Gopalakrishnan, R., Doroudi, S., Ward, A.R., Wierman, A., 2016. Routing and staffing when servers are strategic. Oper. Res. 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. J. Appl. Probab. 916–919.

Hokstad, P., 1978. Approximations for the M/G/m queue. Oper. Res. 26 (3), 510–523.

Kimura, T., 2010. The M/G/s queue. In: Cochran, J.J., Cox, L.A., Keskinocak, P., Kharoufeh, J.P., Smith, J.C. (Eds.), Wiley Encyclopedia of Operations Research and Management Science. John Wiley & Sons, Inc., p. 5.

L'Ecuyer, P., Gustavsson, K., Olsson, L., 2018. Modeling bursts in the arrival process to an emergency call center. In: Proceedings of the 2018 Winter Simulation Conference. In: WSC '18, IEEE Press, pp. 525–536.

Lee, H.L., Cohen, M.A., 1983. A note on the convexity of performance measures of M/M/c queueing systems. J. Appl. Probab. 20 (4), 920–923.

Mandelbaum, A., Stolyar, A.L., 2004. Scheduling flexible servers with convex delay costs: Heavy-traffic optimality of the generalized $c\mu$-rule. Oper. Res. 52 (6), 836–855.

Mehrotra, V., Ross, K., Ryder, G., Zhou, Y.-P., 2012. Routing to manage resolution and waiting time in call centers with heterogeneous servers. Manuf. Serv. Oper. Manage. 14 (1), 66–81.

Orange County Fire Authority, 2014. Standards of Coverage and Deployment Plan. URL: http://www.ocfa.org/Uploads/Orange%20County%20Fire%20Authority%20SOC_FINAL.pdf, Accessed: 2016-09-30.

Pichitlamken, J., Deslauriers, A., L'Ecuyer, P., Avramidis, A.N., 2003. Modelling and simulation of a telephone call center. In: Winter Simulation Conference, Vol. 2. pp. 1805–1812.

Ta, T.A., L'Ecuyer, P., Bastin, F., 2016. Staffing optimization with chance constraints for emergency call centers. In: MOSIM 2016 - 11th International Conference on Modeling, Optimization and Simulation. Montréal, Canada. URL: https://hal.inria.fr/hal-01399507.

Tezcan, T., Dai, J., 2010. Dynamic control of N-systems with many servers: Asymptotic optimality of a static priority policy in heavy traffic. Oper. Res. 58 (1), 94–110.

US Department of Commerce: United States Census Bureau, 2010a. Quick Facts, Irvine city, California. URL: https://www.census.gov/quickfacts/table/POP010210/0636770, Accessed: 2016-09-30.

US Department of Commerce: United States Census Bureau, 2010b. US Census Block Data. URL: https://www.census.gov/geo/maps-data/data/tiger-data.html, Accessed: 2016-09-30.