Reserve driver scheduling
Diwakar Gupta and Fei Li
Industrial & Systems Engineering Department, University of Minnesota, Minneapolis, MN 55455, USA

ABSTRACT
Transit agencies use reserve drivers to cover open work that arises from planned and unplanned time off, equipment breakdowns, weather, and special events. Work assignment decisions must be made sequentially without information about future job requests, a driver’s earlier assignment may not be interrupted to accommodate a new job (no pre-emption), and the scheduler may need to select a particular driver when multiple drivers can perform a job. Motivated by this instance of the interval scheduling problem, we propose a randomized algorithm that carries a performance guarantee relative to the best offline solution and simultaneously performs better than any deterministic algorithm. A key objective of this article is to develop an algorithm that performs well in both average and worst-case scenarios. For this reason, our approach includes discretionary parameters that allow the user to achieve a balance between a myopic approach (accept all jobs that can be scheduled) and a strategic approach (consider accepting only if jobs are longer than a certain threshold). We test our algorithm on data from a large transit agency and show that it performs well relative to the commonly used myopic approach. Although this article is motivated by a transit industry application, the approach we develop is applicable in a whole host of applications involving on-demand-processing of jobs. Supplementary materials are available for this article. Go to the publisher’s online edition of IIE Transactions for datasets, additional tables, detailed proofs, etc.

1. Introduction
Transit agencies (bus, light rail, subway, ferry) use reserve drivers to cover work that arises as a result of planned and unplanned time off, equipment breakdowns, weather, and special events. There are two classes of such work, which we also refer to as jobs. One class contains jobs that are known in advance; e.g., training, union meetings, and vacations. The second class contains those jobs that arise in an online fashion on each day of operations; e.g., open work due to equipment breakdowns, weather events, accidents, drivers calling in sick, and unexpectedly high volume of riders on some routes. Planned open work is assigned to available reserve drivers a day in advance. Unplanned work is assigned as it arises, one piece of work at a time, without knowledge of future pieces of work that may arrive later that day. A dispatcher assigns unplanned open work either to available reserve drivers or to drivers who indicate their willingness to take overtime work assignments, giving rise to the online version of a problem that goes by two names: (i) interval scheduling problem (Kolen et al., 2007; Kovalyov et al., 2007) and (ii) Fixed Job Scheduling (FJS) problem (Gertsbakh and Stern, 1978). This problem is the focus of our article. In particular, we develop an approach for solving the reserve driver scheduling problem that takes into account transit-agency objectives. We begin by describing the institutional background.

1.1. Institutional background
According to DeAnnunzis and Morris (2007), approximately 26% of the total workforce size of large transit agencies consists of reserve drivers. Metro Transit, the agency responsible for the bulk of transit operations in the Twin Cities of Minneapolis and Saint Paul, shared bus driver data with the authors for the period March to August 2010. In this period, Metro Transit operated three large and two small garages and employed approximately 1500 bus drivers, of whom approximately 30% were reserve drivers. Reserve drivers are full time employees with regular 8-hour shifts. They are not assigned duties in advance. Instead, they cover contingent work.

In our data, the average utilization of reserve drivers was between 50 to 60%, depending on the garage (see Gupta et al. (2011) for details). Still, the daily overtime usage during weekdays was well over 100 hours in three of Metro Transit’s large garages. Overtime drivers were paid time and a half (≈$42 per hour). This added tens of thousands of dollars in overtime cost daily. The ensuing description of the reserve driver scheduling problem and the model presented in Section 2 are based on the observations of one of the authors of this article who shadowed a Metro Transit dispatcher for a week.

1. Dispatchers relied on their experience when making dispatch decisions. They had access to software that could...
do offline work assignment, but it was not suited for online assignments.
2. Jobs were sometimes known a few hours before they needed to be performed; e.g., when a driver called in sick several hours before the start of his or her shift. In such cases, dispatchers assigned the job immediately, choosing not to wait until assignments were due.
3. Garages used overtime sign-up sheets. Drivers wrote down the start and end of time periods during which they were willing to do overtime work.
4. In addition to sign-up sheets, dispatchers actively recruited overtime drivers when needed by making phone calls, and they always succeeded in doing so.
5. Dispatchers did not sometimes assign open work to reserve drivers even when such drivers were available. They kept some reserve drivers idle just in case and instead used overtime drivers.
6. Dispatchers did not have any guidance on what to do when multiple reserve drivers were available for a particular assignment and these drivers had different shift end times. Different dispatchers therefore used different heuristic approaches.

In this article we use the total amount of work covered by a given complement of reserve drivers as a measure of performance. This is appropriate and consistent with reality because (i) reserve drivers’ wages are already committed; (ii) overtime drivers are paid for each minute of extra work performed; and (iii) dispatchers are able to cover all work by utilizing overtime workers.

1.2. Literature review

Many papers focus on the online FJS problem and the interval scheduling problem. These papers often use the term processor to refer to the server; i.e., the reserve driver in our setting. Therefore, in the remainder of this article, we use the terms processor and driver interchangeably.

There is a large body of work in which previous assignments may be preempted. This literature is not directly relevant to our setting because once a driver is dispatched to cover an open piece of work, the cost of undoing his or her assignment is prohibitive. We do not provide a detailed review of this body of work. Papers in this stream establish either the Competitive Ratio (CR) or a competitive ratio bound (CB) on the competitive ratio of proposed algorithms under a variety of different problem features; see Woeginger (1994); Faigle and Nawijn (1995); Seiden (1998); Miyazawa and Erlebach (2004); Fung (2008); Fung et al. (2008); Baruah et al. (1992); and Koren and Shasha (1995).

In this article, we build on the work of Lipton and Tomkins (1994), who provide an algorithm that can be used to solve single driver online FJS problem without preemption. This algorithm has a CR of $O((\log \Delta)^{1+\epsilon})$, where $\epsilon$ is any positive real number and $\Delta$ denotes the ratio of the minimum and maximum job durations ($d_{\text{max}}$ and $d_{\text{min}}$, respectively). Lipton and Tomkins identify a converging infinite sequence of coin-flip probabilities that leads to the above CR. Faigle et al. (1996) extend their result in two ways. First, the authors show that multiple different acceptance probability sequences $(f(1), f(2), \cdots)$ can be used to form different randomized algorithms if $\Sigma 1/f(n)$ converges and that a CR of $O(\log(f(n)))$ can be achieved. Second, the article extends the result to include multiple identical processor case.

Differences between these papers and our approach stem from the fact that we develop a customized algorithm for our application domain. First, the upper and lower bounds on job durations are known in our setting. Second, in addition to the worst-case performance guarantee (CR), transit agencies desire good average performance. This is important because an algorithm that has the best performance in terms of the worst case can perform poorly on average. Third, reserve drivers have shift end times, after which they are not available. This means that our algorithm must be able to assign work to one of multiple reserve drivers who can perform that work and whose shifts end at different times while preserving the established CR. We customize our algorithm for this setting, establish a CB, and show using experiments based on actual data that our algorithm also performs well in terms of the average performance criterion.

Next, we contrast our approach with papers that consider multiple processors. Garofalakis et al. (2002) consider the problem of online scheduling of continuous media streams, in which each job requires a fixed processing time and a portion of bandwidth. This can be considered as a model with multiple identical processors and jobs that require different numbers of parallel machines to process. Garofalakis et al. (2002) prove a competitive ratio lower bound of $O((\log \Delta)/(1 - r))$, where $r$ is the maximum fraction of the server’s bandwidth that a job can demand. To accomplish their objective, Garofalakis et al. (2002) divide the available bandwidth (equivalently, the interval $[d_{\text{min}}, d_{\text{max}}]$ in our setting) into $[\log \Delta]$ partitions such that the $\ell$th segment is of length $[2^\ell - 1]d_{\text{min}}, 2^\ell d_{\text{min}})$. When a decision regarding job $j$ needs to be made, if the duration of $j$ lands in the $r$th interval, then it can use at most $i/O((\log \Delta)/(1 - r))$ share of the bandwidth. If job $j$ does not require more than that amount of the bandwidth, then it is accepted; else, it is rejected. We pursue a different approach in this article. We do not determine acceptance probabilities based on a sequence of job duration intervals. Instead, we solve a max–min optimization problem to find the optimal acceptance probabilities.

1.3. Our approach

The objective of this article is to present an implementable solution to the reserve driver scheduling problem. Therefore, our solution strategy is different from those suggested in the literature. Utilizing the marriage problem framework introduced in Lipton and Tomkins (1994) and the fact that the minimum and maximum of job durations are known in the reserve driver scheduling problem, we propose a finite-step marriage problem, whose solution serves as a building block of our approach for solving the single-driver problem scenario. Specifically, its solution consists of optimal coin-flip probabilities that determine whether a job will be assigned to a reserve driver or an overtime driver. We establish the exact CR of the resulting algorithm.

We introduce two parameters in our algorithm. These parameters are denoted as $d_T$ and $\alpha$. The first parameter, $d_T \in (d_{\text{min}}, d_{\text{max}})$, serves to strike a balance between the myopic and strategic approaches. Essentially, the algorithm behaves myopically if a job duration is $d_T$ or longer. The second parameter
$\alpha > 1$ specifies the criterion for considering the next job following a rejection. If the size of the next job is not at least $\alpha$ times that of the previously rejected job, then it is rejected outright. Otherwise, it is considered for assignment based on prescribed coin-flip probabilities. It is straightforward to verify that the suggested algorithm will not be a randomized algorithm if $d_\tau/d_{\min} \leq \alpha$. Therefore, we always choose $d_\tau$ such that $d_\tau/d_{\min} > \alpha$. Previous approaches did not have the parameter $d_\tau$; moreover, they fixed $\alpha = 2$. The introduction of these two parameters allows our algorithm to realize good average performance, in addition to having a guaranteed worst-case performance.

To deal with the problem of allocating a job to one of multiple reserve drivers while preserving the CB established for the single-driver case, we argue that the allocation problem is a special case of the online assignment problem, for which we present a greedy algorithm. We use results available in the literature to argue that at each time epoch, the greedy approach will assign work to at least half of the theoretical maximum number of processors that can be allocated at that time epoch (Korte and Haussmann, 1978). We also show that there cannot be an algorithm in our setting that keeps more than half the theoretical maximum number of processors allocated at any time epoch. Upon combining the results from the two analyses described above, we obtain an approach for solving the multiple-processor online FJS problem with variable shifts that is $\mathcal{O}(\Delta_1 \log \Delta_1)$-competitive, where $\Delta_1 = d_\tau/d_{\min}$ and $\Delta_2 = d_{\max}/d_\tau$. Finally, we report experiments, based on data from Metro Transit, in which we show that our approach performs well on both average and worst-case performance criteria.

The rest of this article is organized as follows. We introduce notation, assumptions, the myopic algorithm, and the finite-step marriage game in Section 2. Then, in Section 3, we analyze the single-driver job scheduling problem, which we refer to as the first subproblem. The second subproblem—i.e., the job-to-processor allocation problem—is analyzed in Section 4. These two parts together comprise our recommended algorithm. Numerical experiments are presented in Section 5 and we conclude the article in Section 6. Formal proofs are included in an Online Supplement.

2. Preliminaries

An arbitrary instance $I$ of the online FJS problem with multiple processors is characterized by a finite set of jobs and processor shifts. Each driver's shift is a time window denoted by $(a_i, b_i)$, where the driver index $i$ is defined such that $a_1 \leq \cdots \leq a_n$. When a driver's shift is cut up because of previous assignments, each available period of the driver is treated as a separate driver with start and end times of the shift equal to the start and end times of the available time window. Driver $i$ cannot accept any job that starts before $a_i$ or ends after $b_i$. Parameters $s_j$ and $e_j$ denote the start and end times of job $j$. Job indices are sorted by their start times (i.e., $s_j \leq s_{j+1}$ for all $j$). The existence of a job is known at $s_j$ and not earlier. The duration of each job $d_j = e_j - s_j$ is its weight or reward. If a job is not assigned to one of the reserved drivers, then it is performed by an overtime driver. It is assumed that there is an ample supply of overtime drivers.

Our online algorithms works as follows. At each job’s arrival epoch, it first selects a candidate driver and then decides whether to accept or reject this job based only on the state of that driver and features of the job. In contrast, an offline algorithm makes decisions after knowing the entire sequence of job requests before making assignment decisions. For an arbitrary randomized algorithm $A$, let $W(A, I)$ denote the random total reward (i.e., the sum of durations of jobs that are processed) realized upon completion of an arbitrary run of $A$ on $I$. Because $W(A, I)$ may vary each time $A$ is applied to the same problem instance $I$, we use $E[W(A, I)]$ to benchmark the performance of $A$. Let $z(I)$ be the value obtained by applying an optimal offline algorithm, which assumes complete information about all jobs in $I$ before scheduling any job. Algorithm $A$ is called $\beta$-competitive if $\beta \times E[W(A, I)] \geq z(I)$ for any instance $I$. The smallest value of $\beta$ such that the above inequality holds for every $I$ is the exact competitive ratio of $A$ (see, e.g., Lipton and Tomkins (1994)).

Because the offline version of our problem is NP-hard (Li and Gupta, 2014), we replace $z(I)$ by an upper bound $U(I)$ and $E[W(A, I)]$ by a lower bound $F(A, I)$. That is, for an algorithm $A$, rather than calculate $CR(A) = \max \{z(I)/E[W(A, I)]\}$, we estimate the upper bound $CB(A) = \max \{U(I)/F(A, I)\} \geq CR(A)$. Our efforts therefore focus on characterizing $U(I)/F(A, I)$ and on designing an algorithm $A$ that results in provably smallest possible value of $CB(A)$, and at the same time performs well on average. We summarize all notation used in this article in Table 1.

To test the average performance, we use the myopic policy that executes all feasible job assignments as a straw man policy. That is, we compare $E[W(A, I)]$ with $E[W(M, I)]$, where $M$ represents a myopic policy, across a whole range of instances $I$. We also compare the performance of our recommended heuristic with that of human dispatchers. These comparisons are reported in Section 5.

| Table 1. Notation. |
|--------------------|
| $n$ | number of processors |
| $i$ | processor index; $i \in [1, \cdots, n]$ |
| $(a_i, b_i)$ | start and end time of the shift of processor $i$, where $a_i < b_i$ |
| $j$ | job index |
| $(s_j, e_j)$ | start and end times of job $j$, where $s_j < e_j$ |
| $d_j'$ | duration (weight) of job $j$; $d_j' = e_j - s_j$ |
| $d_{\min}$ | minimum job duration |
| $d_{\max}$ | maximum job duration |
| $\Delta_1$ | $d_\tau/d_{\min}$ |
| $\Delta_2$ | $d_{\max}/d_\tau$ |
| $\alpha$ | criterion for candidacy of a job when processor is virtually taken |
| $d_t$ | threshold duration; $d_t \in [d_{\min}, d_{\max})$ |
| $\Delta_t$ | $d_t/d_{\min}$ |
| $\delta$ | degree of current virtually taken job |
| $r$ | degree of current job |
| $\gamma_j$ | distance between $S_j$ and $S_{j-1}$ |
| $c_j$ | considered set of processor $j$ |
| $F_j$ | $(c_j \supseteq a_i, e_j \subseteq b_i)$; the set of processors that can perform $j$ |
| $\sigma_j$ | $\max_{c \in c_j} |e_j|$ |
| $t$ | an arbitrary time epoch |
| $J(t)$ | $(s_j \leq t < e_j)$; the set of jobs that cross $t$ |
| $P(t)$ | $(v_j \leq t < b_i)$; the set of processors whose shifts cross $t$ |
| $w(t)$ | the optimal value of bipartite matching problem defined at $t$ |
Before proceeding further, we address two questions that arise at this point. How does our approach relate to the real environment in which the dispatcher operates? Is the result proposed by the algorithm implementable? To answer the first question, let us examine the key assumptions underlying our approach. These are (i) each job’s arrival epoch coincides with its start time; (ii) all jobs in progress may not be preempted by a new arrival; (iii) all rejected jobs are covered by overtime drivers; (iv) a job’s weight is equal to its duration; and (v) the minimum and the maximum job durations are known. In this list, assumptions (ii) to (v) are true in practice. Assumption (i) is an approximation of reality because sometimes the existence of a job may be known prior to its start time (see discussion in the Introduction section). We consider the effect of such departures from our model assumptions when discussing numerical examples. Note that if two or more jobs start at the same time, we assume that either the sequence in which they need to be assigned to drivers is known or is generated randomly. In response to the second question, we will show that our algorithm is easy to implement and could be easily automated. Moreover, via numerical experiments, we also consider the impact of a dispatcher’s ability to overrule recommendations made by our algorithm. The need for such actions may arise because of reasons that may not be easy to model. For example, if all available overtime drivers are inexperienced on a particular route, the dispatcher may use an experienced reserve driver even when our algorithm recommends the use of an overtime driver.

In constructing proofs of various claims throughout this article, we remove all “empty spaces” to simplify our analysis. That is, when there is a period with no demand between two jobs, we remove such time periods and also remove the correspond
cle, we remove all “empty spaces” to simplify our analysis. That
timedriver.

particrour, the dispatcher may use an experienced reserve
may arise because of reasonsthat may not be easy to model. For
mendations made by our algorithm. The need for such actions
consider the impact of a dispatcher’s ability to overrule recom-
will show that our algorithm is easy to implement and could be
easily automated. Moreover, via numerical experiments, we also
consider the impact of a dispatcher’s ability to overrule recom-
mandations made by our algorithm. The need for such actions
may arise because of reasons that may not be easy to model. For
eexample, if all available overtime drivers are inexperienced on
a particular route, the dispatcher may use an experienced reserve
driver even when our algorithm recommends the use of an over-
time driver.

In constructing proofs of various claims throughout this arti-
cle, we remove all “empty spaces” to simplify our analysis. That
is, when there is a period with no demand between two jobs, we
remove such time periods and also remove the correspond-
ing time period from every processor’s shift. Similarly, if a shift
start time $a_i$ is earlier than $z = \min_j \{s_j\}$, the start time of
the first job in $I$, or a shift end time $b_i$ is later than $\bar{z} = \max_j \{e_j\}$, the
finish time of the latest ending job in $I$, then portions of shifts
$(a_i, z)$ and $(\bar{z}, b_i)$ are also removed. Since no jobs can be assigned
during empty spaces caused by either the pattern of arrivals in $I$
or driver shifts, removal of such spaces does not affect the total
work assigned by any algorithm in both online and offline
versions. Such “empty spaces” can be recognized one by one as suc-
cessive jobs are revealed.

### 2.1. The myopic algorithm for single-processor cases

The myopic (greedy) algorithm accepts every job when it arrives
as long as the processor is capable of processing that job. It has
a finite competitive ratio as shown below.

#### Lemma 1. The competitive ratio of the myopic algorithm is $(\Delta + 1)$.

**Proof.** See Section A in the Online Supplement.

The result shown in Lemma 1 is well known. We present its
proof for sake of completeness. The arguments presented in the
proof allow us to establish that $(\Delta + 1)$ is the best (smallest)
CR that any deterministic algorithm can achieve. These argu-
ments are also presented in Section A of the Online Supplement.

Next we present a variant of the marriage problem of Lipton
and Tomkins (1994) that utilizes known duration bounds and
obtains a better CR.

### 2.2. The finite-step marriage problem

In the marriage problem presented in Lipton and Tomkins
(1994), a host selects a number $N$, which the player does
not know. At step $k$, the host offers the player a reward of
$2^k$ for $k = 1, \ldots, N$. At any step, the player can choose to
accept the offer, which terminates the game. If the player
does not accept any offer in all $N$ steps, the game terminates
and the player gets zero. Lipton and Tomkins (1994) pro-
posed a policy, according to which the player would accept at
step $k$ with probability $1/[k^{1+\epsilon}(1+\epsilon)]$, for $k = 1, \ldots, \infty,$
where $\epsilon > 0$. Observe that $\sum_{i=1}^{\infty} 1/(i^{1+\epsilon})$
converges for any positive $\epsilon$, so the policy is well defined.

The finite-step marriage problem is similar to the classic mar-
riage problem introduced in Lipton and Tomkins (1994) with
the difference that there is a maximum possible number of turns
$K \geq 2$ and this fact is known to both the player and the host.
Another difference is that each successive offer is not necessarily
twice as large as the previous one. Rather, it can be any multiple
of $a^k = 1$ of the previous offer. Starting from turn 1, the host offers
the player $a^k$ at turn $k$. The host also picks the number of turns
$N, N < K$. The player does not know $N$ but knows that $N$
cannot be greater than $K$. The player can accept only once. If the player
does not accept in $N$ turns, then the game ends and the player
receives zero.

Suppose the player decides to take the offer at turn $\tilde{N}$. If
$\tilde{N} > 1$, then all those cases in which $N < \tilde{N}$ yield zero reward,
which results in an infinite competitive ratio. If $\tilde{N} = 1$, then
the ratio of the performance relative to the best outcome is
$1/a^{\hat{N}-1}$. That is, the only deterministic strategy that yields a
finite competitive ratio is $\hat{N} = 1$. Choosing to accept at the first turn results in a competitive ratio of $a^{\hat{N}-1}$, which is
bounded by $a^{K-1}$. Next, we investigate whether the player
can earn a greater expected reward by adopting a randomized
strategy.

Suppose the player flips a coin at each turn to decide whether
to accept or reject the host’s offer at that turn. Then, a ran-
donized strategy consists of conditional probabilities $p_k, k =
1, \ldots, K$, of accepting the offer at step $k$, conditioned upon the
event that the game proceeds to step $k$. Lemma 2 identifies an
optimal randomized strategy and we also establish its perform-
ance bound.

#### Lemma 2. The optimal randomized strategy for solving the finite-
step marriage problem consists of coin-flip conditional probabili-
ties $p_1 = a/[K(a-1)+1]$ and $p_k = 1/(K-k+1), k = 2, \ldots, K$.

**This strategy is $[K(a-1)+1]/a$-competitive—i.e., $O(\log \Delta)$-competitive—because $K = \log \Delta + 1$.**

**Proof.** See Section B in the Online Supplement.
3. A randomized algorithm for the single-processor online FJS problem

The proposed algorithm, labeled $A(\alpha, d_T)$, has two discretionary parameters $\alpha$ and $d_T$. Parameter $\alpha > 1$ determines the jobs that are considered by the processor and $d_{\text{min}} < d_T < d_{\text{max}}$ is a threshold duration such that for jobs with duration $d_T$ or greater, algorithm $A(\alpha, d_T)$ uses a greedy approach (accept if feasible), and for jobs with duration less than $d_T$, $A(\alpha, d_T)$ decides based on an approach similar to that in the finite-step marriage game. Thus, $d_T$ determines the extent to which $A(\alpha, d_T)$ is conservative (close to being myopic) or risk taking (close to the marriage problem). It should be clear that $A(\alpha, d_{\text{min}})$ is identical to the myopic approach. We label a job as type 1 if its duration is smaller than $d_T$ and as type 2 if its duration is at least $d_T$.

When job $j$ arrives at time $t$ in an arbitrary instance of single-processor online FJS problem, it may find the processor in one of three states: clear, virtually busy, and busy. Busy indicates that job $j$ overlaps a job that is being processed at $t$. Virtually busy means that job $j$ overlaps a job that was considered but not accepted because it was not long enough and there was an unfavorable coin flip, and clear means neither of the above. A job that causes the driver to be virtually busy is referred to as a virtually taken job.

Algorithm $A(\alpha, d_T)$, upon finding the processor in clear state, will accept the arriving job if its duration is at least $d_T$. Otherwise, this arrival starts the equivalent of a finite-step marriage game. Each such epoch resets a counter that we call the D-level. The value of D-level counter equals $\delta := \lfloor \log_2(d_{\text{max}}/d_j) \rfloor$, where $d_j$ is the duration of the job that starts the marriage game. Next, suppose a job $j$ arrives at time $t$, the processor is virtually busy, and the current D-level is $\delta$. Let $\nu$ denote the index of the virtually taken job at $t$. Define parameter $\ell := \lfloor \log_2(d_{\nu}/d_j) \rfloor$ as the degree of the current virtually taken job. To ascertain whether $d_j$ is sufficiently large relative to $d_{\nu}$, we compare $r := \lfloor \log_2(d_{\nu}/d_j) \rfloor$ and $\ell$. If $r > \ell$, then $d_j$ is deemed sufficiently large and the algorithm would consider job $j$ as a candidate job. This means that it will flip a coin to decide whether to accept or reject $j$. Otherwise, it will reject $j$. The coin-flip probabilities are determined as follows.

If, following $j_1$, each overlapping job is exactly $\alpha$ times as long as its predecessor, then from the marriage problem analogy $j_1$ should be taken with probability $p_1^1 := \alpha/\lfloor \delta(\alpha - 1) + 1 \rfloor$ and each subsequent job with probability $(\alpha - 1)/\lfloor \delta(\alpha - 1) + 1 \rfloor$. (It should be clear that $\delta$ is analogous to $K$ in the finite-step marriage problem and that these probabilities correspond to the unconditional probabilities $c_i$s in Lemma 2.) However, jobs do not necessarily arrive in such a pattern. For example, the job following virtually taken $j_1$ may be $\alpha^2$ times the duration of $j_1$. In such cases, we award the second job the sum of probabilities that correspond to the duration $\alpha d_{j_1}$, $\alpha^2 d_{j_1}$, and $\alpha^3 d_{j_1}$. That is, we calculate the $r$ value to determine which equivalent turn index the new job $j_2$ belongs to and compare it with the turn index of job $\nu$. Note that the turn index in the equivalent finite-step marriage problem is calculated relative to $j_1$ and that the turn index associated with $\nu$ is labeled $\ell$. If $r > \ell$, we calculate the corresponding coin-flip probability (corresponding to conditional probabilities $p_{\ell}$ in Lemma 2). This coin-flip probability is the ratio of the number of the slots from $\ell$ to $r$ divided by the total number of slots from $\ell$ to $\delta$; i.e., $(r - \ell)/(\delta - \ell)$. These arguments lead to Algorithm $A(\alpha, d_T)$ below, which can be implemented in real time.

3.1. The CB

The derivation of the CB hinges upon finding $U(I) \geq z(I)$ and $F(A, I) \leq E[W(A, I)]$. To accomplish this task, we construct disjoint subsets $(S_1, S_2, \ldots)$ of jobs in $I$ that have the following properties. First, the construction allows us to estimate an upper bound on the time period between each $S_k$ and $S_{k+1}$, which we denote by $y_k$ such that $S_k \cup y_k \geq T - \bar{S}$, the relevant span of $I$ (see the discussion just before Section 2.1). We use $|S_k|$ to denote the time period covered by the set $S_k$. Second, by construction, sets $S_k$ have the property that the minimum expected amount of work that $A(\alpha, d_T)$ assigns to the processor within each $S_k$ is independent and identical across all subsets. Therefore, the fraction of work that $A(\alpha, d_T)$ assigns within each $(|S_k| + y_k)$ is no less than $U(I)/F(A, I)$.

The former is then our estimate of CB(A). We begin with the rules for constructing sets $S_k$.

We illustrate the application of these rules with the help of several examples. In Fig. 1, we present an example in which $I$ results in a single chain $S_1$. For this example, we have $\alpha = 2$, $d_T = 4.5$, and five jobs with the following durations: $d_{j_1} = 1$, $d_{j_2} = 2$, $d_{j_3} = 2$, $d_{j_4} = 4$, and $d_{j_5} = 5$. Job $j_1$, being the first job, is automatically included in $S_1$. Next, job $j_2$ is included because $d_{j_2} = \alpha d_{j_1}$. Continuing to apply the rules for composing chains, we find that $j_3$ is not included because $d_{j_3} < \alpha d_{j_2}$, $j_4$ is included because $d_{j_4} = \alpha^2 d_{j_2}$, and, finally, $j_5$ is included because $d_{j_5} > d_T$. There are no other jobs and this terminates the process of forming chains. In this example, job $j_5$ does not belong to a chain.

Next, we present another example in Fig. 2 that explains the construction of $S_k$ after finalizing $S_{k-1}$. The same logic applies to the construction of an arbitrary $S_{k+1}$ following $S_k$. In such cases, the tricky part is finding the “starting job” of $S_k$ because the formation of the chain is identical to that in the earlier example.

![Figure 1](image1.png)

![Figure 2](image2.png)
Algorithm 1: Pseudo-code for Algorithm $A(\alpha, d_T)$

0  when job $j$ arrives at time $t$:
1    if processor is processing a job, then reject $j$
2      end
3    if processor is clear or virtually busy and $j$ is type 2, then accept $j$.
4  end
5  if processor is clear and $j$ is type 1, then calculate the D-level $\delta$ and flip a coin: accept $j$ with probability $p_1^j = \alpha / [\delta(\alpha - 1) + 1]$ or virtually take $j$ with probability $1 - p_1^j$. Set $j_1 = j_r = j$ and $\ell = 0$.
6  end
7  if processor is virtually busy and $d_j < d_T$, then calculate $r = \lceil \log_\alpha (d_j/d_f) \rceil$;
8    if $r \leq \ell$, then reject $j$
9    else flip a coin for job $j$: accept $j$ with probability $(r - \ell)/(\delta - \ell)$ and virtually take $j$ with probability $1 - (r - \ell)/(\delta - \ell)$. Set $j_\ell = j$ and $\ell = r$.
10  end
11  end

Figure 3. The first job of $S_1$ is $j_1$ following Rule 2.2.2.

In Fig. 2, we see that $j_3$ does not qualify as the starting job of $S_k$. Job $j_3$ fails because $d_{j_3} < \alpha d_{j_1}$ and $j_3$ overlaps $S_{k-1}$. Job $j_4$ does not overlap $j_3$ and any other job in $S_{k-1}$. It starts $S_k$. Note that job $j_3$ does not belong to any chain.

Similarly, Fig. 3 shows another example in which job $j_5$ starts $S_k$. In this example, job $j_3$ does not belong to $S_k$ because it overlaps $S_{k-1}$ (Rule 2.2.1). Job $j_4$ overlaps $j_3$, and $d_{j_3} < \alpha d_{j_5}$. Thus, $j_4$ also cannot begin $S_k$ (Rule 2.2.2). However, $d_{j_4} \geq \alpha d_{j_5}$. Therefore, $j_5$ satisfies Rule 2.2.2 and starts $S_k$. Note that jobs $j_3$ and $j_4$ do not belong to any subset.

As seen in examples above, the first job in $S_k$ must be such that during the execution of $A(\alpha, d_T)$, it cannot be rejected on account of its relationship with jobs in $S_k$. Two cases arise. If the last job in $S_k$ has duration less than $d_T$, then the first job $j$ of $S_{k+1}$ must meet the following criterion: either its duration is at least $d_T$ or if its duration is less than $d_T$, then any $j'$ that overlaps $j$ either does not overlap a job in $S_k$ or $\alpha d_{j'} < d_T$. If the duration of the last job in $S_k$ is at least $d_T$, then the first job of $S_{k+1}$ is the first job that does not overlap $S_k$. This method of construction avoids interaction among jobs in $S_k$ and we can analyze the amount of work assigned in each $S_k$ independently.

Lemma 3. The time periods $y_k$ between consecutive pairs $S_k$ and $S_{k+1}$ can be calculated as follows.

Case 1. If all jobs in $S_k$ are type 1, then $y_k$ cannot be greater than $(\alpha^2 + \alpha)|S_k|$.

Case 2. If there is at least one type 2 job in $S_k$, then $y_k$ cannot be greater than $|S_k|$.

Proof. See Section C in the Online Supplement.

Note that the statement in Lemma 3 remains intact for the last set, indexed by $k$. In that case, $y_k$ is defined as the time period between the end of $S_k$ and $T$, the end of $I$. We obtain lower bounds on the expected amount of work assigned in each $S_k$ in the following Lemma.

Lemma 4.

1. If all jobs in $S_k$ are type 1, then the expected amount of work $A(\alpha, d_T)$ will assign on the region covered by $S_k$ is at least $(\alpha - 1)^2 |S_k| / [\alpha^2([\alpha - 1])\log_\alpha \Delta_1 + 1]]$.

Rules for adding a job to $S_k$

1.0 Suppose $j$ is the last-added job in $S_k$. Then add $\hat{j} > j$ to $S_k$ if:

1.1 $j$ overlaps $\hat{j}$ and $d_{\hat{j}} \geq \alpha d_j$

1.2 either $d_{\hat{j}} \geq \alpha d_j$, or $d_j \geq d_T$

Jobs are added to $S_k$ following Rule 1 until there are no qualifying jobs left.

Rules for initiating $S_k$ (finding the first job in $S_k$)

2.0 If $k = 1$, then $j_1$ (the first job in $I$) begins $S_1$ by definition.

Suppose $k \geq 2$, and let $j$ be the last-added job in $S_{k-1}$.

Find the lowest-indexed job $j > j$ to begin $S_k$ such that

2.1 either: $d_j \geq d_T$,

or: $(d_j < d_T)$ and for every job $\ell$ indexed between $j$ and $\hat{j}$ (excluding $j$ and $\hat{j}$) that overlaps $\hat{j}$,

2.2.1 either $\ell$ does not overlap any job in $S_{k-1}$,

2.2.2 or $d_j \geq \alpha d_\ell$. 


2. If there are type 2 jobs in $S_k$, then the expected amount of work $A(\alpha, d_T)$ will assign on the region covered by $S_k$ is at least $\min\{[(\alpha - 1)^2]S_k/\alpha^2[(\alpha - 1)\log_\alpha \Delta_1 + 1], (\alpha - 1)S_k/((\Delta_2 + (\alpha/(\alpha - 1)))[(\alpha - 1)\log_\alpha \Delta_1 + 1])\}$.

Proof. See Section D in the Online Supplement.

From Lemmas 3 and 4 we find that in Case (1), Algorithm $A(\alpha, d_T)$ assigns at least $(\alpha - 1)^2/2\cdot S_k/\alpha^2[(\alpha - 1)\log_\alpha \Delta_1 + 1]$ and the distance between $S_k$ and $S_{k+1}$ is at most $(\alpha^2 + \alpha)/2\cdot S_k$. Similarly, in Case (2), Algorithm $A(\alpha, d_T)$ assigns at least $\min\{[(\alpha - 1)^2]/\alpha^2[(\alpha - 1)\log_\alpha \Delta_1 + 1], (\alpha - 1)/((\Delta_2 + (\alpha/(\alpha - 1)))[(\alpha - 1)\log_\alpha \Delta_1 + 1])\} \cdot S_k$ and the distance between $S_k$ and $S_{k+1}$ is at most $|S_k|$. Upon combining these arguments we obtain our main result in this article, presented in Theorem 1.

Theorem 1. The following is an upper bound of the competitive ratio of $A(\alpha, d_T)$:

$$\kappa \geq \max\left\{ \frac{(\alpha^2 - 1)\alpha^2[(\alpha - 1)\log_\alpha \Delta_1 + 1]}{(\alpha - 1)^2}, \frac{2\alpha^2[(\alpha - 1)\log_\alpha \Delta_1 + 1]}{\alpha - 1}, \frac{2[\Delta_2 + (\alpha/(\alpha - 1))][(\alpha - 1)\log_\alpha \Delta_1 + 1]}{\alpha - 1} \right\}.$$ 

Clearly, Algorithm $A(\alpha, d_T)$ is $O(\Delta_2 \log \Delta_1)$-competitive for the single-processor cases. We investigate how one would choose $\alpha$ and $d_T$ in Section 5 after developing performance bounds for the multiple-processor cases.

Before closing this section, we point out that there may be other algorithms that achieve a competitive ratio bound of $O(\log(\Delta))$. For example, one can divide the duration range in intervals $[2^k d_{\min}, 2^{k+1} d_{\min}]$ and set the acceptance probability for each interval to be $1/(\log_\alpha \Delta)$. Our algorithm is better for two reasons. First, we select acceptance probabilities based on optimizing the finite-step marriage problem. Second, our algorithm is more flexible with parameters $(\alpha, d_T)$, which can be chosen to maximize performance. We use an example next to underscore the importance of using optimal coin-flip probabilities. Assume that job durations occur in lengths of $2^k$, $k = 1, \ldots, K$, all jobs in $I$ overlap, and shorter job always arrive first. Using the framework of a finite-step marriage problem with at most $K$ steps, Lemma 2 provides acceptance probabilities that have a competitive ratio of $\lceil\log \Delta \rceil/2$ upon setting $\alpha = 2$. In contrast, if we were to use uniform acceptance probability of $(1/\log K)$, then the CR would have been $K$. That is, in this example, our approach could provide twice as good worst-case performance.

4. Multiple-processor online FJS problem

Our recommended algorithm for dealing with problem instances involving multiple processors is denoted by $A_{\alpha}(\alpha, d_T)$, where $n$ is the number of processors. $A_{\alpha}(\alpha, d_T)$ utilizes an "allocate and assign" approach. The allocate portion of this approach determines the processor that will be asked to consider the job. The assign part is the Algorithm $A(\alpha, d_T)$ discussed in Section 3. It determines whether a processor will accept a job that is allocated to that processor. Note that each job is allocated to only one processor. It is not re-allocated, even if the job is rejected because it overlapped an in-progress job at that processor.

In this section, we focus on the allocation part, which we refer to as Procedure G. We show that at every observation epoch, Procedure G will guarantee that at least half of the theoretical maximum number of processors that can be allocated a job are indeed allocated a job. Upon combining these results and those from the previous section, we argue that $A_{\alpha}(\alpha, d_T)$ retains the order of magnitude of the CB established in Section 3.

Let $F(j) = \{a_i \leq s_j < e_j \leq b_j\}$ be the set of processors that can perform job $j$. For each processor, we keep track of the set of allocated jobs. This set is denoted by $C_i$. In addition, we define $\tau_i$ as the end time of the last-arriving job in $C_i$, i.e., $\tau_i = \max_{j \in C_i} \{e_j\}$. Here allocated means that a job is routed to processor $i$ but it may or may not be accepted. The decision to accept the job is made after the allocation decision. The act of allocating a job $j$ to processor $i$ may make it necessary to update $\tau_i$, because $j$ may now be the last-arriving job in $C_i$.

Procedure G selects a processor in $F(j)$ that has the smallest $\tau$ value among processors in $F(j)$. Every job is allocated by G because every job can be processed by at least one processor and G does not consider potential overlap with previously allocated jobs when allocating the next job to a processor.

Allocation Procedure G

Step 0: when job $j$ arrives at time $t$, identify $F(j) = \{a_i \leq s_j < e_j \leq b_j\}$. Let $F(j) = \{a_i \leq s_j < e_j \leq b_j\}$.

Step 1: find $i = \arg\min_{k \in F(j)} \tau_k$, allocate job $j$ to processor $i$ and update $\tau_i$.

Before presenting formal arguments, we informally explain our main result. Suppose that all jobs are revealed to the dispatcher and that he or she counts the number of jobs and the number of driver shifts available at $t$, for every $t$. Only considering this set of jobs and this set of processors, we designate a job-to-processor pair as a "feasible match" if that job's interval lies within the driver's shift. Then, we solve a bipartite matching problem to determine $w(t)$, the largest possible number of matches. If we integrate $w(t)$ over all $t$, we will obtain an upper bound on the total amount of work that may be covered by available reserve drivers. Note that it is generally not possible to make $w(t)$ matches at every $t$ because jobs may overlap. Let $w_C(t)$ denote the number of allocations made by G. Next, we argue that Procedure G identifies at least $w(t)/2$ matches at every $t$ (i.e., $w_C(t) \geq w(t)/2$). This argument is based on an earlier result in the literature. The integral of $w_C(t)$ over all $t$ is the amount of job time allocated to reserve drivers, which is at least half of the integral of $w(t)$. Combining this result with the CR obtained from $A(\alpha, d_T)$, we obtain the overall CR.

For an arbitrary time $t \in [0, T]$, we define $I(t) = \{a_i \leq s_j < e_j \leq b_j\}$ as the set of jobs that need processing at $t$ and $P(t) = \{a_i \leq s_j < e_j \leq b_j\}$ as the set of processors that are available at $t$. We associate a node with each job in $I(t)$ and with each processor in $P(t)$. Then we connect two nodes, one associated with a job $j$ and the other with a processor $i$, with an undirected edge if and only if job $j$ can fit in processor $i$'s shift; i.e., if $a_i \leq s_j < e_j \leq b_i$. This exercise gives rise to a bipartite graph. The relevant matching problem is
to find the maximum number of unconnected edges. The maximum matching problem thus defined is related to the job-to-processor allocation problem as shown in Lemma 5.

**Lemma 5.** For each $t$, let $w(t)$ be the number of matched pairs in an optimal solution to the maximum matching problem. Then, in any job-to-processor allocation procedure, the number of processors that are allocated at least one job at time $t$ cannot be greater than $w(t)$.

**Proof.** See Section E in the Online Supplement.

**Lemma 6.** The number of processors allocated by Procedure G at any time $t$ is at least $w(t)/2$.

**Proof.** See Section F in the Online Supplement. We describe next a problem instance in which Procedure G allocates exactly $w(t)/2$ jobs for some $t$. This shows that $w(t)/2$ is a tight bound on the performance of Procedure G.

Suppose there are two processors, labeled $P_1$ and $P_2$, with shift start and end times $(0, 10)$ and $(4, 14)$, and three jobs, labeled $j_1, i \in \{1, 2, 3\}$, with start and end times $(1, 6)$, $(4, 9)$, and $(7, 12)$. Note that job $j_1$ can be allocated only to processor $P_1$ and job $j_3$ only to processor $P_2$. It should be clear that upon executing Procedure G, $j_1$ will be allocated to $P_1$, and $j_2$ and $j_3$ to $P_2$. Next, we observe that at $t = 8$, job $j_2$ can be matched with $P_1$ and $j_3$ with $P_3$. That is, $w(8)$ is two but Procedure G allocates only $P_1$ at $t = 8$.

**Lemma 7.** No allocation procedure can guarantee to allocate jobs to more than $w(t)/2$ processors at every $t$ for all problem instances.

**Proof.** See Section G in the Online Supplement. Using Lemma 6, we next establish the overall competitive ratio of our approach in Theorem 2.

**Theorem 2.** An upper bound of the competitive ratio of $A_{\alpha}(\alpha, d_T)$ is

\[
2 \times \max \left\{ \frac{\alpha^3 - 1}{\alpha^2} \left[ (\alpha - 1) \log_\alpha \Delta_1 + 1 \right], \frac{2 \alpha^2 (\alpha - 1) \log_\alpha \Delta_1 + 1}{(\alpha - 1)^2}, \frac{2 \Delta_2 + (\alpha/(\alpha - 1)) \log_\alpha \Delta_1 + 1}{\alpha - 1} \right\}.
\]

### 5. Numerical experiments

We obtained data concerning reserve driver operations for five randomly picked months from Metro Transit. The agency had five garages and each month’s data came from a different garage. The data included all jobs that were assigned to either reserve or overtime drivers. The agency did not drop service in the period for which we obtained the data, which meant that the jobs in our data set represented all open work. We considered each day as an independent instance and excluded weekends from our experiments because weekends have a special service schedule. This resulted in approximately 100 problem instances, one for each weekday of operations (five garages $\times$ ($\approx$ 20 days per garage).

The shortest job was approximately half an hour long, and the longest job possible was 8 hours long, with a few exceptions. These exceptions include cases when absentee drivers worked a non-standard shift (either 9- or 10-hour shift) or when there were very short pieces of work. As the exceptions are rare, we set $\Delta = 16$ in our experiments. Figure 8(a) in the Online Supplement H shows a box plot of job durations by garage. We see that Garage 1 and Garage 5 have jobs with more variable durations and that Garage 3’s job durations do not vary as much. The available time of each driver is a continuous 8-hour period, but drivers start at different time of day, in order to cover the entire period during which buses operate (20 hours in each 24 hour period). The shift start and end times are read from data.

Box plots in Figs. 8(b) and 9 in the Online Supplement H show the variation in problem size by garage. The daily number of reserve drivers ranged from 7 to 18 and the daily number of jobs ranged from 8 to 36. The job-to-driver ratio also varied significantly across garages. We graphed the demand profile by time of day for Garages 1 and 3 in Fig. 10 in the Online Supplement H and found differences in the pattern of job arrival times. Figure 10 was obtained by dividing time into 15-minute intervals and counting the number of jobs that were active in each interval across all weekdays for each garage. We observed that every garage experienced two peaks each day, but Garage 3’s peaks were sharper. Pattern differences observed in Figs. 8 to 10 in the Online Supplement H influence the performance of our algorithm, which we explain later in this section.

For comparing performance of our algorithm, we chose the myopic approach as the straw-man policy. There are several reasons for doing so. First, the myopic approach is the most commonly used approach in practice. Second, in addition to ensuring that work assignment algorithms have bounded worst-case performance, transit agency managers are also interested in achieving good average performance. Myopic or greedy algorithms generally produce a good average performance and in some cases, it can be proved that their worst-case performance is also good; e.g., in the maximum coverage problem (Feige, 1998), the greedy algorithm achieves theoretically best worst-case performance (Cornuejols et al., 1977). Finally, the myopic approach is the most conservative policy that accepts jobs in their arrival sequence so long as a feasible assignment is possible. Our algorithm contains a parameter $d_T$ that provides a hedge between myopic and strategic approaches. We also compare the performance of our algorithm with the performance of the human dispatchers. The latter is calculated from the data.

For each garage, we searched for the best parameters by comparing the results for different combinations of $(\alpha, d_T)$, where $\alpha \in [1.1, 2.4]$ (in increments of 0.1) and $d_T \in [90, 240]$ (in increments of 10 minutes). We used two different search criteria. In one case, $\alpha$ and $d_T$ were chosen to maximize the worst-case performance of the randomized algorithm and in the other case to maximize its average performance. Note that selected parameters vary by garage, which is caused by differences in patterns of job durations, frequency, and driver availability as explained in Figs. 8 to 10 in the Online Supplement H. Also, the performance of optimal values $(\alpha^*, d^*_T)$ was generally not significantly different from other values of $\alpha$ and $d_T$ that were
close to the optimal values. Both sets of results are reported in Table 2.

For each day, we simulated 10 runs of $A_\alpha(\alpha, d_T)$ to obtain an estimate of its expected performance. We report the percentage of total job time that each algorithm assigns averaged over 20 days of data for each garage. Columns 2 to 5 show comparisons when parameters of the randomized algorithm were selected to maximize worst-case performance and columns 6 to 9 show similar results for average performance. Finally, columns 10 and 11 show the average advantage from the randomized algorithm in terms of minutes of overtime saved daily. We observe that as compared with the myopic algorithm, the use of the randomized algorithm could save a total of 1091 minutes, or 18.2 hours, of overtime daily. At approximately $42 per hour in overtime wages, this could save approximately $190,925 annually (assuming 250 work days in a year). The savings are even greater when we compare with the human dispatcher.

We point out that the actual process of work assignment is more complex than the randomized algorithm. For example, sometimes dispatchers will know about an open piece of work a day before and tentatively assign that piece to an overtime driver. The following day, they may change the assignment to a reserve driver. Also, in some cases, some pieces of work are assigned to overtime drivers when they sign up. We suspect that this is done to make it attractive for drivers to sign up for overtime. As such procedures are ad hoc, comparisons between that this is done to make it attractive for drivers to sign up for overtime. As such procedures are ad hoc, comparisons between the performance of rule-based algorithms and human dispatchers’ performance may exhibit a great deal of variability across garages. For example, human dispatchers perform slightly better than myopic algorithm for Garage 1 but significantly worse for Garage 3. However, in all cases, our algorithm outperforms both the myopic algorithm and the human dispatchers on both the worst-case and the average performance criteria. We study the impact of dispatcher overrule in Section 5.1.

A transit agency may optimize parameters ($\alpha, d_T$) either for the worst-case or the average performance but not both. It is therefore important to understand the sensitivity of the randomized algorithm’s performance to selected parameters. In the test results reported in Table 3, we compare the average and worst-case performances when parameters are optimized for the worst-case and the average performances, respectively. We observe that if parameters that are optimal for the worst-case performance are used, the average performance deteriorates between 0.8 and 1.9%. Similarly, when parameters are optimized for the average performance, the worst-case performance deteriorates between 0.5 and 1.9%. The analysis confirms that the algorithm’s performance is reasonably stable with respect to the criterion used to obtain optimal ($\alpha, d_T$).

In addition to experiments based on actual service days, we also simulated problem instances by sampling from the set of all jobs and driver shifts in our data set. There were three reasons for performing such experiments. First, we could in this way generate many more than 100 problem instances. Second, we could design experiments with different problem sizes; i.e., numbers of jobs and drivers. Third, we could test the performance of the algorithm in a realistic setting by selecting parameters $\alpha$ and $d_T$ based on a training data set and testing the realized performance over randomly generated test data.

Problem instances in our experiments have different job-to-driver ratios but preserve the ranges we observed in the data. Specifically, we observed that the daily minimum, median, and maximum numbers of drivers in the 100 problem instances were 8, 13, and 17. Similarly, the daily minimum, median, and maximum job-to-driver ratios were 1.0, 1.6, and 2.3. Therefore, our simulated instances have nine different combinations. These are eight processors with 8, 13, or 18 jobs; 13 processors with 13, 21, or 30 jobs and 17 processors with 17, 27, or 39 jobs (see Tables 4 and 5).

Tables 8 and 9 in the Online Supplement I show a summary of simulated data. We consider each day as one sample that gives rise to that day’s average job duration and average shift start

### Table 2. Performance comparison—myopic (displayed as “MP”) and human dispatcher (displayed as “HM”) versus $A_\alpha(\alpha, d_T)$.

| Garage number | Worst case | Average | Daily advantage |
|---------------|------------|---------|-----------------|
|               | $D_T = (\alpha, d_T)$ | $D_T = (\alpha, d_T)$ | $D_T = (\alpha, d_T)$ |
| \hline
| 1 | 21.1 | 18.7 | (1.7, 210) | 35.0 | 50.5 | 52.3 | (1.5, 200) | 53.6 | 285 | 120 |
| 2 | 51.8 | 39.6 | (1.8, 140) | 57.5 | 68.5 | 60.1 | (1.9, 200) | 70.6 | 167 | 835 |
| 3 | 48.8 | 26.3 | (1.4, 110) | 53.8 | 69.4 | 36.0 | (1.6, 210) | 71.8 | 110 | 1641 |
| 4 | 38.1 | 36.4 | (1.3, 120) | 48.0 | 65.0 | 60.7 | (1.5, 170) | 69.7 | 206 | 394 |
| 5 | 42.8 | 34.8 | (1.4, 210) | 46.0 | 61.3 | 56.6 | (1.9, 240) | 65.0 | 323 | 733 |

| Note: Performance = percentage of total work assigned. |

### Table 3. Performance sensitivity to parameters ($\alpha, d_T$).

| Garage number | \hline
| 1 | (1.7, 210) | 35.0 | 52.2 | (1.5, 200) | 33.9 | 53.6 |
| 2 | (1.8, 140) | 57.5 | 69.8 | (1.9, 200) | 55.6 | 70.6 |
| 3 | (1.4, 110) | 53.8 | 70.5 | (1.6, 210) | 53.3 | 71.8 |
| 4 | (1.3, 120) | 48.0 | 67.8 | (1.5, 170) | 46.7 | 69.7 |
| 5 | (1.4, 210) | 46.0 | 63.5 | (1.9, 240) | 44.5 | 65.0 |

Note: Performance = percentage of total work assigned.

### Table 4. Parameter selection using training data set.

| Jobs, Performance, \hline
| 8, 8 | 56.1 (16.8) | (1.5, 160) | 59.8 (19.6) |
| 13, 8 | 45.6 (17.5) | (1.4, 230) | 54.2 (22.4) |
| 18, 8 | 36.8 (13.8) | (1.6, 230) | 50.2 (18.2) |
| 13, 13 | 60.8 (24.1) | (1.4, 230) | 63.6 (24.6) |
| 21, 13 | 47.4 (23.3) | (1.1, 140) | 61.7 (36.0) |
| 30, 13 | 36.2 (13.9) | (1.2, 210) | 53.9 (27.8) |
| 17, 17 | 60.5 (27.8) | (1.1, 190) | 66.8 (31.8) |
| 27, 17 | 48.5 (19.5) | (1.9, 230) | 63.2 (32.1) |
| 39, 17 | 37.6 (18.5) | (1.9, 120) | 59.0 (32.9) |

Note: Performance = percentage of total work assigned.
time. Then, treating each of these averages as a single observation, we calculated the summary statistics in Tables 8 and 9. To explain these statistics further, we discuss two examples. In Table 8, the minimum average duration of 66 means that among 300 days of simulated data with eight jobs, the minimum daily average duration was 66 minutes. Similarly, in Table 9, among 300 simulated days with eight drivers, the minimum daily average shift start time was 5:01 am.

We generated 300 random instances for each set of parameters independently as training and test data. Parameters \( \alpha \) and \( d_T \) were selected to maximize randomized algorithm's worst-case performance on the training data. The same values, denoted by \( \alpha^* \) and \( d_T^* \), were then applied to the test data. Similar to Table 2, this involved searching for best \( \alpha \) and \( d_T \) over a range of possible values (\( \alpha \in [1.1, 2.4] \) in increments of 0.1 and \( d_T \in [90, 240] \) in increments of 10). The selected parameters provided the best worst-case performance (which is shown in parentheses in Tables 4 and 5), and the average performance was based on the same parameters. For each problem instance, we simulated \( A_n(\alpha^*, d_T^*) \) 10 times to estimate the expected performance. The number of repetitions was limited because of computational burden and because expected performance was quite stable over 10 simulations.

Table 5 shows that \( A_n(\alpha^*, d_T^*) \) performs better on both worst-case and average performance criteria in all problem instances relative to the myopic approach. The relative advantage of our approach, which we call Daily Advantage in Table 5, was measured as average percentage of extra work assigned and as additional average minutes of work assigned on each day of operations with each set of parameters. For transit agencies with large garages (i.e., with many jobs and drivers), the randomized algorithm has the potential to save significant overtime costs. The performance of both algorithms in experiments involving the simulated data is not as good as with real data because in the simulated data, drivers' shift schedules were determined by a random pick. In contrast, dispatchers assigned start times a day before each day of operations and they were able to better match shift starts to the time-of-day pattern of job arrivals in their respective garages.

Table 5 suggests that the job-to-driver ratio affects relative performance. As expected, the average percentage of work assigned decreases as the job-to-driver ratio increases. However, generally more work is assigned when there are more drivers and concomitantly more jobs. This is not surprising, as with more jobs and more drivers, there are more opportunities to benefit from the use of randomized algorithm.

### 5.1. Implementability

As mentioned in Section 2, if our algorithm were implemented, it would be important to allow the dispatcher to overrule the algorithm's recommendation based on certain factors that might be difficult to incorporate into a formal model. Moreover, dispatchers may assign a piece of work as soon as it becomes known, rather than wait until it is due. For these reasons, we consider the effects of dispatcher overrule and early assignment of certain jobs in this section. In these experiments, we use real data and the values of \( (\alpha^*, d_T^*) \) obtained in Table 2.

#### 5.1.1. Dispatcher overrule

The experiments reported in this section were performed using real data from the five garages. We ran our algorithm in a normal fashion but allowed the dispatcher to overrule; i.e., take a course that is opposite of what the algorithm recommends. Specifically, this means that the dispatcher will reject a job (assign to overtime) if the algorithm's decision is assign and assign if the algorithm's decision is reject. Note that we allow overrule only when the algorithm uses coin-flips to determine whether to accept a job or not. That is, we do not consider overrule for jobs that are greater than \( d_T \) in length.

We considered 5%, 10%, and 15% chances of overruling the algorithm's recommendation. As overruling is permitted only when the job length is less than \( d_T \), an experiment with 10% overrule probability does not imply that 10% of decisions pertaining to all jobs can be overruled. A summary of our results is provided in Table 6, which should be considered the analog of Table 2 after allowing overruling. Note that the reported performance metric is the percentage of total job durations that is assigned to reserve drivers (i.e., that does not incur overtime charges). From Table 6 we see that across all five garages, performance decreases between 0 and 2.5% for the worst case and between 0 and 3.4% for the average case, as compared with the result of \( A_n(\alpha^*, d_T^*) \) with no overruling. Specifically, for a 5% chance of overruling, the ranges are [0%, 1%] for the worst case and [0%, 1%] for the average case; for a 10% overrule, the ranges are [0%, 2.3%] for the worst case and [0.3%, 2.9%] for the average case; and for a 15% overrule, the ranges are [0.5%, 2.3%] for the worst case and [2.3%, 3.1%] for the average case.

#### 5.1.2. Early assignment

We also consider the situation when some jobs are known in advance and the dispatcher immediately assigns those jobs to
Table 7. Performance comparison—with the chance of early assignment.

| Garage number | Worst case 10% | Average 10% | \( A_\ast (\alpha^\ast, d_T^\ast) \) Worst, avg |
|---------------|---------------|-------------|---------------------------------------------|
| 1             | 31.0          | 49.9        | 35.0, 53.6                                 |
| 2             | 51.3          | 63.3        | 57.5, 70.6                                 |
| 3             | 47.3          | 64.5        | 53.8, 71.8                                 |
| 4             | 43.5          | 62.8        | 48.0, 69.7                                 |
| 5             | 41.8          | 58.5        | 46.0, 65.0                                 |

Note: Performance = percentage of total work assigned to reserve drivers.

Although the focus of this article on algorithm development, it offers some generalizable insights. First, by considering only those jobs whose lengths are increasing in a geometric sequence, the randomized algorithm leads to worst-case performance that is of order \( \log \Delta \). (Recall that \( \Delta = d_{\text{max}}/d_{\text{min}} \).) This implies that the performance of our algorithm does not substantially degrade when job durations cover a wide range. In contrast, the best deterministic algorithm has performance of order \( \Delta \), which does degrade linearly. This establishes that the desirability of using randomized algorithms increases with \( \Delta \). Second, \( \Delta \) affects the likelihood of finding a value of \( \alpha \) that makes the virtual algorithm more competitive than the myopic approach. The larger the value of \( \Delta \), the larger this likelihood. However, with a relatively large value of \( \alpha \), a randomized algorithm takes more risk (rejects more jobs) and could perform poorly on average. This risk can be counterbalanced by introducing a parameter such as \( d_T \). That is, randomized algorithms, which choose optimal \( \alpha \) and \( d_T \), are likely to result in desirable outcomes for other highly random scheduling environments as well.

Acknowledgments

The authors are grateful to the IIE Transactions’ review team and Departmental Editor Professor Sanjay Mehrotra for helping us improve the manuscript.

Funding

This material is based upon work supported in part by the National Science Foundation under grant no. CMMI-1332680. Furthermore, this material is also based upon work completed, in part, while Diwakar Gupta was on detail at the National Science Foundation under the terms of an Intergovernmental Personnel Assignment, Award no. CMMI-1452877. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Notes on contributors

Diwakar Gupta is a Professor of Industrial & Systems Engineering at the University of Minnesota. He is also an affiliate senior member of the Health Services Research Policy and Administration Division of the School of Public Health and a Faculty Scholar of the Center for Transportation Studies. At the time of writing of this bio sketch, he is on assignment to the National Science Foundation, where he is serving as a Program Director for the Service, Manufacturing and Operations Research program. He received a Ph.D. in Management Sciences from the University of Waterloo. His research focuses on healthcare delivery systems, state transportation agencies’ operations, and supply chain and revenue management. Information about his research projects can be found at the web page of the Supply Chain and Operations Research Laboratory (www.isye.umn.edu/labs/scorlab).

Fei Li received his Ph.D. in Industrial & Systems Engineering from the University of Minnesota. The research reported in this article is a part of his Ph.D. dissertation. He is currently employed at the Hennepin County Medical Center as a Performance Improvement Advisor. In this capacity he is developing models, tools, and solutions for different levels of staff scheduling decisions.

References

Baruah, S., Koren, G., Mao, B., Mishra, B., Raghunathan, A., Rosier, L. and Shasha, D. (1992) On the competitiveness of on-line real-time task scheduling. The Journal of Real-time Systems, 4(2), 125–144.
Cornuejols, G., Fisher, M.L. and Nemhauser, G.L. (1977) Location of bank accounts to optimize float: an analytic study exact and approximate algorithms. Management Science, 23, 789–810.

DeAnnuntis, C.P. and Morris, W.P. (2007) Transit extraboard management—optimum sizing and strategies final report. Available at: http://www.nctr.usf.edu/pdf/77707.pdf Accessed September 20, 2014.

Faigle, U., Garbe, R. and Kern, W. (1996) Randomized online algorithms for maximizing busy time interval scheduling. Computing, 56(2), 95–104.

Faigle, U. and Nawijn, W.M. (1995) Note on scheduling intervals on-line. Discrete Applied Mathematics, 58(1), 13–17.

Feige, U. (1998) A threshold of ln n for approximating set cover. Journal of the ACM, 45(4), 634–652.

Fung, S.P.Y. (2008) Lower bounds on online deadline scheduling with pre-emption penalties. Information Processing Letters, 108(4), 214–218.

Fung, S.P.Y., Poon, C.K. and Zheng, F. (2008) Online interval scheduling: randomized and multiprocessor cases. Journal of Combinatorial Optimization, 16(3), 248–262.

Garofalakis, M., Ioannidis, Y., Özden, B. and Silberschatz, A. (2002) Competitive on-line scheduling of continuous-media streams. Journal of Computer and System Sciences, 64(2), 219–248.

Gertsbakh, I. and Stern, H.I. (1978) Minimal resources for fixed and variable job schedules. Operations Research, 26(1), 68–85.

Gupta, D., Li, F. and Wilson, N. (2011) Extraboard-driver workforce planning for bus transit operations. CURA Reporter, 41(3–4), 11–18.

Kolen, A., Lenstra, J., Papadimitriou, C. and Spieksma, F. (2007) Interval scheduling: a survey. Naval Research Logistics, 54(5), 530–543.

Koren, G. and Shasha, D. (1995) An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. SIAM Journal on Computing, 24, 318–339.

Korte, B. and Hausmann, D. (1978) An analysis of the greedy algorithm for independence systems. Annals of Discrete Mathematics, 2, 65–74.

Kovalyov, M., Ng, C. and Cheng, T. (2007) Fixed interval scheduling: models, applications, computational complexity and algorithms. European Journal of Operational Research, 178(2), 331–342.

Li, F. and Gupta, D. (2014) The extraboard operator scheduling and work assignment problem. IIE Transactions, 46(11), 1132–1146.

Lipton, R.J. and Tomkins, A. (1994) Online interval scheduling, in Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, Association for Computing Machinery, New York, and the Society for Industrial and Applied Mathematics, Philadelphia, pp. 302–311.

Miyazawa, H. and Erlebach, T. (2004) An improved randomized online algorithm for a weighted interval selection problem. Journal of Scheduling, 7(4), 293–311.

Seiden, S. (1998) Randomized online interval scheduling. Operations Research Letters, 22, 171–177.

Woeginger, G.J. (1994) Online scheduling of jobs with fixed start and end times. Theoretical Computer Science, 130(1), 5–16.