Online Supplement for Reserve Driver Scheduling

by Diwakar Gupta and Fei Li

A Proof of Lemma 1

We assume without loss of generality that $d_{\min} = 1$ and therefore $\Delta = d_{\max}$. We will establish two claims:

(1) $(\Delta + 1)$ is an upper bound on the competitive ratio, that is, the competitive ratio is no more than $(\Delta + 1)$; and

(2) that for any $\epsilon > 0$, there exists an instance in which the myopic algorithm assigns no more than $(1 + \epsilon)/(\Delta + 1)$ of the optimal assignment – that is, the competitive ratio is no less than $(\Delta + 1)/(1 + \epsilon)$.

This means the competitive ratio lies in the range $((\Delta + 1)/(1 + \epsilon), \Delta + 1)$. Furthermore, because ϵ can be arbitrarily small, these claims together prove the statement of Lemma 1.

After empty spaces are removed, if the dispatcher uses a myopic algorithm, then the arrival of a job when the driver is idle must start a busy period because the myopic algorithm will accept that job. Because $d_{\max} = \Delta$ and idle periods under a myopic algorithm result from job overlap, all idle periods must be shorter than Δ . If we take any two connected busy and idle periods, then their total duration is at most $(d + \Delta)$, where $d \geq 1$ is the duration of the accepted job. No algorithm can assign more than $(d + \Delta)$ amount of work in each pair of connected busy and idle periods. Therefore, the largest CB in this instance is $(d + \Delta)/d$, which is maximized upon setting $d = 1$. This proves the first claim.

For the second claim, we draw attention to an instance of the problem illustrated in Figure 4, where $\epsilon > 0$ is arbitrary. Job 1: $(0, 1 + \epsilon)$, Job 2: $(\epsilon/2, 1 + \epsilon/2)$, and Job 3: $(1 + \epsilon/2, 1 + \Delta + \epsilon/2)$. In this instance, the myopic algorithm results in the driver being busy for the duration $(1 + \epsilon)$.

Figure 4: A $(\Delta + 1)$ -Competitive Case

Jobs j_2 and j_3 overlap j_1 and cannot be taken after j_1 is taken. But the optimal solution is to take jobs j_2 and j_3 with total duration $(\Delta + 1)$. This proves the second claim. $\square$

The arguments presented in the proof of Lemma 1 can be used to establish that $(\Delta + 1)$ is the best (smallest) competitive ratio that *any* deterministic algorithm can achieve. We believe such arguments are well known, but we include them here for sake of completeness. Every deterministic algorithm can be placed into one of two classes when a job with duration $(d_{\min} + \epsilon)$ arrives, and the processor is idle – Class 1 contains algorithms that accept that job and Class 2 contains algorithms that reject it. Next, we compare their performance in two instances: (1) the example shown in Figure 4, and (2) the example in which the job with duration $(d_{\min} + \epsilon)$ is the only job. We see that algorithms in Class 1 by accepting the first job have a competitive ratio of $(\Delta + 1)$, whereas Class-2 algorithms by rejecting the first job have a competitive ratio of ∞ , establishing that the best competitive ratio must be $(\Delta + 1)$.

B Proof of Lemma 2

Let c_k denote the unconditional probability that the player accepts the offer at step k . Clearly, p_k and c_k are related as follows: $p_k = c_k / (\sum_{i=k}^K c_i)$, $k = 1, \dots, K$. The problem of finding p_k 's is equivalent to that of finding a set of probabilities $\{c_1, \dots, c_K\}$ satisfying $\sum_k c_k = 1$. The performance ratio if $N = k$ is $R_k := \sum_{i=1}^k \alpha^{i-k} c_i$ such that $R_1 = c_1$ and $R_k = R_{k-1}/\alpha + c_k$, for $k = 2, \dots, K$. The player would choose c_k 's to maximize the worst-case performance, i.e. solve $\max[\min\{R_1, \dots, R_K\}]$, subject to $\sum_k c_k = 1$, and $c_k \geq 0$. This is achieved by setting all R_k 's equal, i.e. $R_k = \alpha/[K(\alpha - 1) + 1]$ for every k , which gives $c_1 = \alpha/[K(\alpha - 1) + 1]$ and $c_k = (\alpha - 1)/[K(\alpha - 1) + 1]$ for $i = 2, \dots, K$. Such a strategy leads to a competitive ratio of $[K(\alpha - 1) + 1]/\alpha$. Given c_k 's, it is easy to calculate the coin-flip probabilities shown in the statement of the lemma. Hence proved. $\square$

C Proof of Lemma 3

Our proof for Case (1) follows the logic presented in Lipton and Tomkins (1994) but generalizes it for an arbitrary $\alpha > 1$ and known duration bounds (Lipton and Tomkins 1994 considers only the $\alpha = 2$ case). No job that overlaps with S_k can be longer than $\alpha|S_k|$ because otherwise it should belong to S_k . Therefore, any job starting outside S_k with duration at least $\alpha^2|S_k|$ qualifies to be the first job of S_{k+1} . In the extreme case,

655 a job with duration $\alpha|S_k| - \epsilon$ overlaps S_k , and another job with duration $\alpha^2|S_k| - \epsilon$ overlaps the previous job,
 656 giving rise to the maximum y_k of $(\alpha + \alpha^2)|S_k|$ – an example is shown in Figure 5. A type-2 job also qualifies
 657 to be the starting job of S_{k+1} , but in that case, y_k is smaller leaving intact the worst-case argument presented
 658 above.

Figure 5: Largest y_k (when all jobs in S_k are type-1)

659 In Case (2), S_k contains at least one type-2 job and we select the first job that does not overlap any job
 660 that overlaps S_k as the first job of S_{k+1} . The worst case in this situation is one in which a job with duration
 661 $(d_T - \epsilon)$ overlaps both S_k and S_{k+1} . In this case y_k is at most $(|S_k| - \epsilon)$, which happens when S_k contains
 662 only one job with duration d_T . □

663 D Proof of Lemma 4

664 The proof of Case (1) uses arguments similar to those presented in Lipton and Tomkins (1994), which
 665 considers the $\alpha = 2$ case. We provide a proof with arbitrary α for sake of completeness.

666 Let j_m be the last job of an arbitrary subset S_k and let σ be a feasible schedule of work assignments.
 667 Because in this case, all jobs are of type 1, it follows that $d_{j_m} < d_T$. For an arbitrary run of the algorithm
 668 $A(\alpha, d_T)$, when presented with j_m , one of the following cases may occur.

669 (a) processor is clear;

670 (b) processor is busy processing a job labeled j_ℓ , and either

671 (b.1) $d_{j_m} < \alpha d_{j_\ell}$, or

672 (b.2) $d_{j_m} \geq \alpha d_{j_\ell}$;

673 (c) processor is virtually processing a job labeled j_ℓ , and either

(c.1) $d_{j_m} \geq \alpha d_{j_\ell}$, or

(c.2) $d_{j_m} < \alpha d_{j_\ell}$.

In Case (a), $A(\alpha, d_T)$ will flip a coin for j_m with success probability $p_1^\delta = \alpha/[(\alpha - 1)\delta + 1]$. That is, j_m is assigned with probability at least $\alpha/[(\alpha - 1)\delta + 1]$.

In Case (b.1), j_ℓ whose duration is at least $1/\alpha$ times the duration of j_m , would have been taken with probability at least $(\alpha - 1)/[\delta(\alpha - 1) + 1]$.

To evaluate Case (b.2), we consider all jobs that arrive within (s_{j_ℓ}, s_{j_m}) , i.e. the interval between the start times of jobs j_ℓ and j_m , and use $\bar{S}$ to denote this set. If there exist some jobs in $\bar{S}$ with duration greater than $(1/\alpha)d_{j_m}$, we denote the set of such jobs by $\bar{S}'$. If $\bar{S}'$ is not empty, then there exists at least one job j in the set $\bar{S}' \cup \{j_m\}$ with the property: $\lfloor \log_\alpha(d_j/d_{j_1}) \rfloor > \lfloor \log_\alpha(d_{j_\ell}/d_{j_1}) \rfloor$, where j_1 is the index of the job that started the finite-step marriage problem sequence associated with j_ℓ . Pick the first job satisfying this property in the set $\bar{S}' \cup \{j_m\}$ and refer to it as j'_ℓ . Then, j'_ℓ , with duration at least $(1/\alpha)d_{j_m}$, would be taken with probability at least $(\alpha - 1)/[\delta(\alpha - 1) + 1]$. Alternatively, $\bar{S}'$ may be empty because all jobs in S_ℓ have duration less than $(1/\alpha)d_{j_m}$. Then j_m would be taken with probability at least $(\alpha - 1)/[\delta(\alpha - 1) + 1]$. These two cases are shown in Figures 6 and 7.

Figure 6: When no job in $\bar{S}$ has duration $(1/\alpha)d_{j_m}$ or larger.

In Case (c.1), a coin flip would decide if j_m is accepted. That is, j_m would be taken with probability at least $(\alpha - 1)/[\delta(\alpha - 1) + 1]$. In Case (c.2), a coin was flipped (unsuccessfully) at an earlier job arrival epoch resulting in j_ℓ being virtually taken. Job j_ℓ could have been taken with probability at least $(\alpha - 1)/[\delta(\alpha - 1) + 1]$. Note that d_{j_ℓ} is at least $(1/\alpha)d_{j_m}$.

Now we consider the fact that $|S_k| \leq d_{j_m}(1 + \alpha^{-1} + \alpha^{-2} + \dots) < (\frac{\alpha}{\alpha-1})d_{j_m}$, which means $d_{j_m} > (\frac{\alpha-1}{\alpha})|S_k|$. Putting together the above arguments, we have shown that a job of duration at least $(1/\alpha)(\frac{\alpha-1}{\alpha})|S_k|$ would

Figure 7: When at least one job in $\bar{S}$ has duration $(1/\alpha)d_{j_m}$ or larger (j_3 in this case).

be accepted with probability $(\alpha - 1)/[\delta(\alpha - 1) + 1]$. That is, a lower bound of expected work performed is $(\alpha - 1)^2|S_k|/\{\alpha^2[\delta(\alpha - 1) + 1]\}$.

For Case (2), let j denote the first type-2 job in S_k . Then, j divides S_k into two parts: every job before j is of type-1 and every job starting with j is type-2. Two cases now arise:

(2.1) If all type-2 jobs overlap a type-1 job, we have $|S_k| < (\Delta_2 + \frac{\alpha}{\alpha-1})d_j$. This is the worst case in which $d_j = d_T$, job j overlaps the longest-possible job within the set of type-1 jobs, and another job with $d_{\max} = \Delta_2 d_j$ also overlaps job j . So the part of S_k that contains type-1 jobs cannot be longer than $\frac{\alpha}{\alpha-1}(d_T - \epsilon)$, and the part that contains type-2 jobs cannot be longer than $\Delta_2 d_j$.

The probability that j is accepted equals the probability that all type-1 jobs are virtually taken. This probability is at least $(\alpha - 1)/[(\alpha - 1)\log_\alpha \Delta_1 + 1]$. Therefore, the expected assigned time in this case is at least $(\alpha - 1)|S_k|/\{(\Delta_2 + \frac{\alpha}{\alpha-1})[(\alpha - 1)\log_\alpha \Delta_1 + 1]\}$.

(2.2) If not all type-2 jobs overlap a type-1 job, then the myopic approach kicks in as soon as $A(\alpha, d_T)$ encounters its first type-2 job that can be assigned. Whether the type-1 job that overlaps a type-2 job is accepted (not virtually) depends on the random realization of $A(\alpha, d_T)$, but we know for sure that either the first type-2 job that overlaps type-1 job will be accepted, or the first type-2 job that does not overlap a type-1 job will be accepted. After $A(\alpha, d_T)$ accepts the first type-2 job, the algorithm will accept every non-overlapping job that is of duration at least d_T in a myopic fashion. This implies that it yields a $\frac{1}{\Delta_2+1}$ worst-case performance on that part of S_k which is covered by type-2 jobs. On the part covered by type-1 jobs, the algorithm still assigns at least $(\alpha - 1)^2/\{\alpha^2[(\alpha - 1)\log_\alpha \Delta_1 + 1]\}$ of the covered time. It is easy to see that $\frac{1}{\Delta_2+1}$ is greater than $(\alpha - 1)/[(\Delta_2 + \frac{\alpha}{\alpha-1})[(\alpha - 1)\log_\alpha \Delta_1 + 1]]$,

with the assumption that $\Delta_1 > \alpha > 1$. Hence proved. $\square$

E Proof of Lemma 5

Because we focus on time t , the relevant sets of jobs and processors are those that belong to $J(t)$ and $P(t)$, respectively. For a given job-to-processor allocation, assume $m'(t)$ processors are allocated at least one job. Then, by choosing one allocated job from each of the $m'(t)$ allocated processors, we can find $m'(t)$ job-to-processor pairs, and this corresponds to an $m'(t)$ -matching. Because $w(t)$ is the optimal value of the bipartite matching problem, it immediately follows that $m'(t) \leq w(t)$. $\square$

F Proof of Lemma 6

Let $m'(t)$ denote the number of processors that are allocated at least one job at t . Procedure G chooses a feasible processor with the smallest τ when a new job arrives. This implies that this procedure always selects an unmatched processor that connects to the current job whenever an unmatched processor exists. Then, the statement of the Lemma follows from the well-known result that a greedy algorithm that picks a new edge (excluding those already selected), achieves a performance bound of at least $1/2$ (Korte and Hausmann 1978). If there are no unmatched processors at t , then $m'(t) = w(t)$, completing the proof. $\square$

G Proof of Lemma 7

Consider time points 5 and 8 in the same problem instance we described just before Lemma 7. It is easy to see that $w(5) = w(8) = 2$. Given that j_1 can be allocated only to P_1 and j_3 only to P_3 , if a procedure allocates j_2 to P_1 , then $m'(5) = 1$, whereas if it allocates j_2 to P_2 , then $m'(8) = 1$. That is, no allocation procedure in this problem instance can assign work to more than half of $w(t)$ processors simultaneously for both time points 5 and 8. This completes the proof. $\square$

H Summary of Metro Transit Data

(a) Job Durations by Garage

(b) Daily Number of Jobs by Garage

Figure 8: Job Durations and Daily Frequency

(a) Driver Numbers by Garage

(b) Job-to-Driver Ratio by Garage

Figure 9: Daily Driver Availability and Job-to-Driver Ratio

(a) Garage 1

(b) Garage 3

Figure 10: Demand Profile by Time of Day

I Summary of Simulated Data

Table 8: Job Duration Summary (minutes), SD= Standard Deviation

# of Jobs	Min	Max	Mean	SD
8	66	340	175	55
13	80	307	175	39
17	90	290	173	34
18	98	314	172	34
21	91	293	173	35
27	84	247	171	29
30	98	286	173	36
39	88	307	172	34

Table 9: Driver Shift Start Time Summary

# of Drivers	Min	Max	Mean
8	5:01 AM	12:34 PM	8:52 AM
13	5:10 AM	12:41 PM	8:56 AM
17	6:08 AM	12:18 PM	9:00 AM