Lower Bounds for Two Call Control Problems

Andrew Tomkins*

October 28, 1995

Abstract

We present lower bounds for two call control scheduling problems. For the problem of scheduling infinite-duration calls deterministically without pre-emption on a linear array of processors, in which the goal is to use as many edges as possible, we show a $2\phi + 1 \approx 4.2$ lower bound on the competitive ratio (where ϕ is the golden ratio). For the problem of scheduling infinite-duration calls on an edge of capacity 1 with randomization but no pre-emption, such that calls have bandwidth between $1/B$ and 1, we show a $(\log B)/2$ lower bound.

Keywords: Algorithms, Call Control, Scheduling

1 Introduction

The *call admission problem* is the problem of determining whether an incoming call should be accepted to a communication network, or whether it should be rejected in order to preserve resources for future calls. A call admission algorithm must accept or reject each call in an on-line manner, *i.e.*, before seeing any future calls. Algorithms are evaluated using the *competitive ratio*, defined by Sleator and Tarjan [5] to be the maximum over all input sequences of the benefit attained by the optimal solution divided by the benefit attained by the algorithm.

Garay, Gopal, Kutten, Mansour, and Yung [4] study call admission for linear array networks of processors. They define various models depending on whether calls may be pre-empted, whether algorithms may use randomization, and how much benefit algorithms accrue for scheduling a particular call. They present a deterministic algorithm for scheduling with pre-emption, in which the goal is to maximize the total number of network edges in use. Thus, this model favors calls between distant processors on the line. They show that, with respect to this benefit measure, the competitive ratio of their algorithm is $2\phi + 1 \approx 4.2$, where ϕ is the golden ratio, $\frac{1+\sqrt{5}}{2}$. They also adapt a lower bound of [3] to show

*andrewt@cs.cmu.edu

that no algorithm can achieve a competitive ratio better than 4. We show that no algorithm in this model can achieve a competitive ratio better than $2\phi + 1$.

Additionally we consider an algorithm of Awerbuch, Bartal, Fiat and Rosen [1] for scheduling on n-node trees with randomization but no pre-emption. The goal is to maximize the total benefit. Edges are unit capacity, and calls are parametrized by start point, end point, duration (between 1 and T), bandwidth (between $1/B$ and 1), and benefit (between 1 and F). They give an algorithm with competitive ratio $O(\log n \log T \log B \log F)$ and a lower bound of $\Omega(\log n \log T \log F)$. We give a lower bound of $(\log B)/2$ on a single edge, which can be cascaded with the other lower bounds to match the upper bound (to within a constant). The authors of [1] have a similar bound using different techniques [2], which is expected to appear in the journal version of their paper.

The problem we solve may also be phrased as follows: let S be a set of identical items. An algorithm must accept or reject requests of the form "Give me k items from S." The goal of the algorithm is to accept as many requests as possible. The techniques of [1] show how to design such an algorithm; we show that the competitive ratio of any algorithm is at least $(\log |S|)/2$.

2 Scheduling on a Linear Array

We present the lower bound for scheduling on an n-node linear array. In this model, a deterministic algorithm must schedule infinite-duration calls with pre-emption. The benefit of each call is the length of the call (*i.e.*, the number of edges of the linear array traversed). The benefit of a schedule is the sum of the benefits of all calls in place at the end of the schedule, so pre-empted calls contribute zero benefit.

Theorem 1 *No algorithm for scheduling in this model can attain competitive ratio better than* $2\phi + 1$.

Proof: At any point during the operation of a scheduling algorithm, various intervals of the line (calls) will have been presented to the algorithm, some rejected, some accepted, and some accepted and then pre-empted. For any such state of the algorithm, a new interval with a particular start point and length will be either accepted or rejected. As the interval increases in length, the algorithm will eventually have to accept the interval (unless $1/(2\phi + 1)$ of the line has already been scheduled). If the interval is very short, but one or more long intervals would have to be pre-empted to schedule the interval, presumably the algorithm would not choose to schedule it. So for any (left or right) start point, we can identify a certain minimal length that an interval must achieve at which point the algorithm will schedule it. We call this length the *critical length* of the interval. It will be clear from context whether the interval is meant to grow to the right or to the left. To be more formal, we define the critical length of an interval I with a fixed start point and direction, with respect to algorithm A, as follows:

$$\text{Critical Length}(I) = \inf\{l \in \mathrm{R}^+ | A \text{ takes } I \text{ if } |I| = l\}.$$

Now consider a sequence of intervals as follows. The first interval has length 1. The next interval starts at the rightmost point of the first interval (with a tiny overlap), and extends to the right for the critical length of the interval. The third interval starts just overlapping the rightmost point of the second interval, and extends to its critical length, and so on. Such a sequence is shown in Figure 1. Clearly the algorithm will take each interval, pre-empting the previous one.

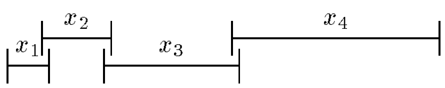

Figure 1: A sequence of intervals

Lemma 1 *If the critical length of each interval is at most ϕ times the length of the previous interval then the algorithm is at most $2\phi + 1$-competitive.*

Proof: We assume that there are at least $\log_\phi(n)/2$ intervals in the sequence; we will justify this assumption later. The algorithm must be competitive on every prefix of the entire sequence. Further, if the adversary halts the sequence at some interval x_j, he can reduce the length of x_j by a tiny amount such that the algorithm does not take pre-empt x_{j-1} to take x_j. Thus, our approach is to show that for some j, $\sum_{i=1}^{j} |x_i| \geq (2\phi + 1)|x_{j-1}|$; this will prove the lemma.

Break the $\log_\phi(n)/2$ intervals into $l = \sqrt{\log_\phi(n)/2}$ batches of l intervals per batch. For each i, find the interval of batch i whose length is the greatest multiple of the previous interval; let this multiple be r_i, and call the interval I_i. By the conditions of the lemma, we must have $0 < r_i \leq \phi$. Thus, by pigeonholing, there must be some i such that $r_i < r_{i+1} + \phi/l$. Find such an i, and renormalize such that the interval immediately before I_{i+1} has length 1, and I_{i+1} has length $r = r_{i+1}$.

Since all l intervals before the length-1 interval must have length at most $r + \phi/l$ greater than the interval before them, the total length to the left of I_{i+1} must be at least the following:

$$1 + \frac{1}{r + \phi/l} + \frac{1}{(r + \phi/l)^2} + \cdots + \frac{1}{(r + \phi/l)^l} = \frac{\left(\frac{1}{r+\phi/l}\right)^{l+1} - 1}{\frac{1}{r+\phi/l} - 1}$$

$$= \frac{r}{r-1} - \frac{\phi/l + (r + \phi/l)^{-(l+1)}(r + \phi/l)(r-1)}{(r + \phi/l - 1)(r-1)}.$$

We set ϵ to be the latter term. Assume first that $r < 1.3$. Since $r_i < r_{i+1} + \phi/l$, for sufficiently large l, if $r < 1.3$ then $r_i < (2\phi+1)/(2\phi) \approx 1.309$. The total length of the l intervals to the left of I_{i+1} must therefore be at least $\max\{r, r_i\}/(\max\{r, r_i\} - 1) > 2\phi + 1$. Since the last element of that subsequence has length 1, the lemma is certainly true. We may therefore assume that $r \geq 1.3$. Since r is bounded away from 1, we note that as l grows large, $\epsilon \to 0$.

Since the length of I_{i+1} is exactly r, the total length of the sequence through I_{i+1} is at least $r/(r-1) - \epsilon + r = r^2/(r-1) - \epsilon$. Since

$$\frac{d}{dr}\left(\frac{r^2}{r-1}\right) = \frac{r(r-2)}{(r-1)^2},$$

which is negative for $1 < r < 2$, we conclude that for $1 < r \leq \phi$,

$$\frac{r^2}{r-1} \geq \frac{\phi^2}{\phi-1} = 2\phi + 1.$$

The competitive ratio of the algorithm is the total length of the sequence divided by the length of the scheduled interval. Since the scheduled interval has length 1, the competitive ratio is therefore bounded by:

$$\text{C.R.} \geq 2\phi + 1 - \epsilon \to 2\phi + 1.$$

□

Thus, if the critical length of an interval is never more than ϕ greater than that of the previous interval, the theorem holds. We must now consider the case in which the critical length is greater than ϕ for some interval in the sequence. We proceed as follows. Since the algorithm has taken a request of length 1 and refused another overlapping request of length $\phi + \epsilon$ to the right, it must take a request beginning at the left of the length-1 interval, and extending $\phi - \epsilon$ to the left, in order to remain competitive. Figure 2 shows three such intervals; if the algorithm does not take the rightmost interval, it must take the leftmost one. Now, however, the algorithm is in the position of having taken an interval of length $\phi - \epsilon$ when $1 + \phi + \epsilon$ of intervals to the right have been offered and either rejected or pre-empted. So there is another length interval we can offer to the left of the current interval that the algorithm must take in order to remain competitive. We shall show a sequence of intervals we can offer to the algorithm extending to the left such that the algorithm must always pre-empt and take the interval offered, and the intervals offered eventually grow sufficiently slowly that the algorithm cannot remain competitive by either taking or leaving the offered interval.

Assume that at some point in the construction of Figure 1, the critical length of some interval is more than ϕ greater than the length of the previous interval. Offer an interval just less than the critical length, which the algorithm will reject. Normalize lengths so that the previous interval has length 1. Then this

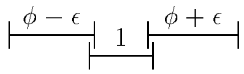

Figure 2: Three Intervals

new interval has length $y > \phi$. Now offer another interval beginning at the other side (say, the left side) of the length-1 interval, extending leftwards for length $x_1 = 2\phi - y$, so that the algorithm must accept x_1. We then offer another interval starting at the left of x_1, with length x_2, the minimal length that the algorithm must take. Next, x_3 starts to the left of x_2, and so on. We show an example of such a sequence in Figure 3. We can write a series of equations that must be satisfied for the algorithm to remain $2\phi+1$-competitive, as follows:

$$\begin{aligned} x_1 + y + 1 &= 2\phi + 1 \\ \frac{x_2 + x_1 + y + 1}{x_1} &= 2\phi + 1 \\ \frac{x_3 + x_2 + x_1 + y + 1}{x_2} &= 2\phi + 1 \\ &\vdots \end{aligned}$$

Rewriting:

$$\begin{aligned} x_2 + 2\phi + 1 &= x_1(2\phi + 1) \\ x_3 + (2\phi + 1)x_1 &= x_2(2\phi + 1) \\ &\vdots \end{aligned}$$

Which in general gives $x_{n+1} = (2\phi + 1)(x_n - x_{n-1})$. We now examine the ratios of successive lengths in this series:

Lemma 2 *If* $\frac{x_n}{x_{n-1}} \leq \phi - \epsilon$ *then* $\frac{x_{n+1}}{x_n} \leq \max\{\phi - \epsilon, 1.53\}$ *for* $0 < \epsilon < \phi - 1$.

Proof: We distinguish two cases. First, we show that for

$$\epsilon \leq (1 + \phi - \sqrt{\phi^4 - 4/\phi})/2 \approx .26,$$

$x_{n+1}/x_n \leq \phi - \epsilon$. Then we show that for $\epsilon > .26$, $x_{n+1}/x_n \leq 1.53$.

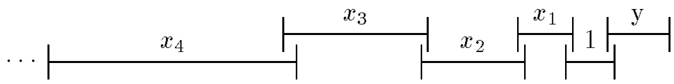

Figure 3: Sequence of Left-Growing Intervals

$$\text{For } \epsilon \leq \frac{1+\phi-\sqrt{\phi^4-4/\phi}}{2} \approx .26$$

$$\begin{aligned}
\frac{x_{n+1}}{x_n} &= (2\phi+1)(1-\frac{x_{n-1}}{x_n}) \\
&\leq \phi^3\left(1-\frac{1}{\phi-\epsilon}\right) \\
&= \phi^3\left(\frac{\phi-1-\epsilon}{\phi-\epsilon}\right) \\
&= \frac{\phi^3}{1+\epsilon-\epsilon^2}\left(2-\epsilon^2-\phi\right) \\
&\leq \frac{\phi^3}{1+\epsilon-\epsilon^2}(2-\phi) \\
&= \frac{\phi}{1+\epsilon-\epsilon^2} \\
&\leq \phi-\epsilon \quad (\text{for } \epsilon \leq .26)\,.
\end{aligned}$$

$$\text{For } \epsilon > \frac{1+\phi-\sqrt{\phi^4-4/\phi}}{2} \approx .26$$

$$\begin{aligned}
\frac{x_n}{x_{n-1}} &\leq \phi-\epsilon = 1.36-\alpha, \alpha \geq 0 \\
\frac{x_{n+1}}{x_n} &= (2\phi+1)(1-\frac{x_{n-1}}{x_n}) \\
&\leq \phi^3\left(1-\frac{1}{1.36-\alpha}\right) \\
&= \phi^3\left(\frac{.36-\alpha}{1.36-\alpha}\right) \\
&\quad (\rightarrow \alpha \leq .36) \\
&\leq \phi^3(.36-\alpha) \\
&\leq \phi^3(.36) < 1.53
\end{aligned}$$

□

Thus, by applying Lemma 1, we can see that in this case also, the competitive ratio cannot be better than $2\phi+1$.

We have overlooked one detail. In order for an off-line algorithm to achieve the performance described above, we imagine making the following change to the request sequence: whenever an interval is taken, we break it into $2\phi+1$ non-overlapping parts and offer each one in turn to the algorithm. The deterministic algorithm cannot accept any of them, or the performance will be automatically equal to the lower bound, as the algorithm must have rejected or pre-empted all intervals except the current one. But the off-line algorithm, knowing that the larger interval will not be taken, can accept all the smaller intervals getting an amount of length equal to the total amount either accepted or accepted then pre-empted by the original algorithm. Our proofs above showed that this amount is what we require to show the lower bound.

We note that these sequences all contain roughly $\log_\phi(n)$ elements before the requests cover the entire line. In summary, our proof proceeds as follows: begin in the middle of the line, growing a sequence to the right. If we can always offer intervals that grow by less than ϕ, say for $\log_\phi(n)/2$ iterations, then Lemma 1 will apply. If at some point we can offer an interval that grows by more than ϕ then we offer an interval just smaller than this one, and begin a new sequence growing to the left, forcing the algorithm to pre-empt at each step. We show that these new intervals grow by less than ϕ and must always be taken, again for about $\log_\phi(n)/2$ iterations, so we apply Lemma 1 once more to complete the theorem. □

3 Scheduling on an Edge

We now present the lower bound for scheduling on a unit capacity edge with bandwidth between $1/B$ and 1. In this model, a randomized algorithm must accept or reject requests for bandwidth on the edge. Accepted calls will remain in place forever, and may not be pre-empted. The total bandwidth of these calls must not exceed 1. Each call has unit benefit, so the benefit of the algorithm on a sequence is the number of calls accepted from the sequence.

Theorem 2 *No algorithm for scheduling in this model can attain competitive ratio better than* $(\log B)/2$*.*

Proof: We show the lower bound using an application of Yao's minimax theorem [6]. We wish to show that there will be a difficult sequence for any randomized algorithm. Instead, by applying the minimax theorem, we show that there exists a distribution over a set of request sequences such that any deterministic algorithm will have low expected competitive ratio.

More precisely, in terms of game theory, we define the *value* of the game played between a particular deterministic algorithm and a particular fixed sequence to be the reciprocal of the competitive ratio: the number of calls scheduled by the algorithm divided by the number of calls scheduled optimally. For any deterministic algorithm A and sequence σ we write the value as $V(A, \sigma)$.

Let R be a randomized algorithm, R_{det} be a deterministic algorithm induced by the random decisions of R, and σ be a fixed sequence. We define γ_R to be the random variable defined by $V(R_{\text{det}}, \sigma)$, over the distribution induced by the coin flips of R. We will show that the expectation of the value is

$$E[\gamma_R] \leq \frac{2}{\log B}.$$

This will suffice to show the theorem because the reciprocal function is convex; thus, by Jensen's inequality, we have:

$$E\left[\frac{1}{\gamma_R}\right] \geq \frac{1}{E[\gamma_R]}.$$

Thus, if C.R. is the competitive ratio, or the reciprocal of γ_R:

$$E[\text{C.R.}] = E\left[\frac{1}{\gamma_R}\right] \geq \frac{1}{E[\gamma_R]} \geq \frac{\log B}{2}.$$

Therefore, it remains to prove that for any randomized algorithm R, $E[\gamma_R] \leq \frac{2}{\log B}$.

We now present a family of sequences to be used in the proof; afterwards, we will present a distribution over these sequences.

Let sequence σ_i, for $i \in [1..\log B]$, be defined by the following program:

Program for σ_i:
For $k \leftarrow 1$ to i do
    Submit 2^k calls, each with bandwidth 2^{-k}.

Thus, σ_1 contains two requests, each for calls of bandwidth $1/2$. σ_3 contains two requests of bandwidth $1/2$, then four requests of bandwidth $1/4$, then eight requests of bandwidth $1/8$. The distribution over these sequences is uniform: with probability $1/\log B$ we choose σ_i.

A deterministic algorithm A whose input is one of the sequences σ_i can be characterized completely by a set of values $c_1 \ldots c_{\log B}$. Each c_i represents the number of calls of size 2^{-i} taken by the algorithm, assuming that the sequence contains calls of that size. In order to be a valid algorithm, the total bandwidth along the edge cannot exceed 1:

$$\sum_{i=1}^{\log B} c_i 2^{-i} \leq 1 \tag{1}$$

Since the optimal algorithm will accept 2^i calls, and deterministic algorithm A will accept $\sum_{j=1}^{i} c_j$ calls, we can write the value of the algorithm as follows:

$$V(A, \sigma_i) = \frac{\sum_{j=1}^{i} c_j}{2^i}.$$

We can now write the expectation of the value when algorithm A is presented with the uniform distribution over the sequences σ_i:

$$\begin{aligned} E[V(A, \sigma_i)] &= \sum_{k=1}^{\log B} \frac{1}{\log B} V(A, \sigma_k) \\ &= \frac{1}{\log B} \sum_{k=1}^{\log B} \frac{\sum_{j=1}^{k} c_j}{2^k} \\ &= \frac{1}{\log B} \sum_{k=1}^{\log B} c_k \left(\frac{2}{2^k} - \frac{1}{B} \right) \\ &= \frac{2}{\log B} \sum_{k=1}^{\log B} \frac{c_k}{2^k} - \frac{1}{\log B} \sum_{k=1}^{\log B} \frac{c_k}{n}. \end{aligned}$$

By the minimax theorem, the expectation of the value, given above, is equal to the value of the best randomized algorithm on a worst-case sequence. We wish to show that the competitive ratio is at least $\log B/2$; thus, we show that the value is at most $2/\log B$. To show this result by contradiction, we assume that the value $2/\log B$ is in fact achieved by algorithm A above:

$$E[V(A, \sigma_i)] \geq \frac{2}{\log B}.$$

And from above:

$$E[V(A,\sigma_i)] = \frac{2}{\log B}\sum_{k=1}^{\log B}\frac{c_k}{2^k} - \frac{1}{\log B}\sum_{k=1}^{\log B}\frac{c_k}{n}.$$

Therefore,

$$\begin{aligned}
\frac{2}{\log B}\sum_{k=1}^{\log B}\frac{c_k}{2^k} - \frac{1}{\log B}\sum_{k=1}^{\log B}\frac{c_k}{n} &\geq \frac{2}{\log B}\\
\sum_{k=1}^{\log B}\frac{c_k}{2^k} - \frac{1}{2}\sum_{k=1}^{\log B}\frac{c_k}{n} &\geq 1\\
\sum_{k=1}^{\log B}\frac{c_k}{2^k} &> 1.
\end{aligned}$$

But this result contradicts Equation 1, which gives our contradiction and shows the theorem. □

4 Acknowledgements

Thanks to Merrick Furst and Avrim Blum for many helpful comments and support. Thanks also to the authors of [1] for helpful discussions of the status of the bandwidth lower bound and to Baruch Schieber for helpful comments on an earlier draft of the pre-emptive lower bound. Finally, thanks to an anonymous referee for several helpful suggestions, clarifications and simplifications.

References

[1] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. In *Symposium on Discrete Algorithms*, 1994.

[2] Y. Bartal. Personal communication, 1995.

[3] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Sasha, and F. Wang. On the competitiveness of on-line real-time task scheduling. In *Proc. IEEE Real-Time Systems Symposium*, pages 106–115, 1991.

[4] J. Garay, I. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control algorithms. In *2nd Annual Israel Conference of Computing and Systems, Netania, Israel*, 1993.

[5] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[6] Andrew Yao. Towards a unified measure of complexity. In *Proceedings of the 18th Annual IEEE Symposium on Foundations of Computer Science*, pages 222–227, 1977.