THEORY OF COMPUTING, Volume 1 (2005), pp. 105–117
http://theoryofcomputing.org

Combining Online Algorithms for Acceptance and Rejection

Yossi Azar* Avrim Blum† David P. Bunde‡ Yishay Mansour§

Received: January 21, 2005; published: September 14, 2005.

Abstract: Resource allocation and admission control are critical tasks in a communication network that often must be performed online. Algorithms for these types of problems have been considered both under benefit models (e.g., with a goal of approximately maximizing the number of requests accepted) and under cost models (e.g., with a goal of approximately minimizing the number of requests rejected). Unfortunately, algorithms designed for these two measures can often be quite different, even polar opposites. In this work we consider the problem of combining algorithms designed for each of these objectives in a way that is good under both measures simultaneously. More formally, we are given an algorithm A that is c_A competitive with respect to the number of accepted requests and an algorithm R that is c_R competitive with respect to the number of rejected requests. We show how to derive a combined algorithm with competitive ratio $O(c_R c_A)$ for rejection and $O(c_A)$ for acceptance. We also build on known techniques to show that given a collection of k algorithms, we can

*Research supported in part by the Israel Science Foundation and by the IST Program of the EU.
†Research supported in part by NSF grants CCR-0105488, CCR-0122581, ITR IIS-0121678 and by a BSF grant.
‡Research supported in part by NSF grant CCR-0093348.
§Research supported in part by the Israel Science Foundation (grant No. 1079/04) and by a BSF grant.

ACM Classification: F.2.2, C.2.2

AMS Classification: 68W05

Key words and phrases: Algorithms, communication networks, resource allocation, online algorithms, competitive analysis, Quality of Service, admission control

 DOI: 10.4086/toc.2005.v001a006

construct one master algorithm that performs similarly to the best algorithm among the k for the acceptance problem and another master algorithm that performs similarly to the best algorithm among the k for the rejection problem. Using our main result we can combine the two master algorithms to a single algorithm that guarantees both rejection and acceptance competitiveness.

1 Introduction

Resource allocation is one of the most critical tasks in communication networks. Many network resources are in constant "short supply": this includes bandwidth (of the various links), queuing delays (or rather the lack of queuing delays in the switches), the ability to route with bounded jitter, and many more. If one would like to guarantee Quality of Service (QoS), one needs to allocate resources to the requesting calls, and since those resources are bounded, it implies that requests must be rejected due to the lack of sufficient resources in certain cases. A simple example is bandwidth allocation. Suppose we have a link with a given capacity, and different calls request bandwidth allocation on that link. Since the system cannot allocate more than the link capacity, it may be forced to reject some of the requests.

The resource allocation (or admission control) decision must typically be done online. That is, the algorithm has to decide for each request whether or not to accept that request (and grant it the resources) while having minimal (or no) knowledge of future requests. This leads very naturally to the setting of online algorithms and using competitive analysis to evaluate performance. In fact, a wide range of resource allocation problems have been considered in this general online setting, including call control, admission control, active queue management, and switch throughput.

When one applies competitive analysis, one needs to decide what performance measure to focus on. One can try either to minimize the number of rejected requests or alternatively to maximize the number of accepted requests. We say that an algorithm is *c-reject-competitive* if it rejects at most c times the number of requests rejected by the optimal algorithm OPT and *c-accept-competitive* if it accepts at least $1/c$ times as many requests as OPT. Even though OPT simultaneously both maximizes the number of accepted requests and minimizes the number of rejected requests, it is a well known phenomena in approximation and online algorithms that approximation ratios are not preserved when considering the two complementary problems. In the literature the minimization version is referred to as a *cost problem* and the maximization version is referred to as a *benefit problem*. There might be one algorithm A that achieves a good ratio for maximizing benefit but has a poor ratio in terms of cost, and a different algorithm R that has a good ratio for minimizing cost but a poor ratio in terms of benefit. For example, consider a 2-accept-competitive algorithm A and a 2-reject-competitive algorithm R. If the optimal solution OPT accepts 98% of the input, algorithm R accepts at least 96%, but algorithm A may accept only 49%. However, if OPT accepts 50% of the input, algorithm A accepts at least 25%, but algorithm R may accept nothing.

In the offline setting, the fact that we have two different algorithms, one for each measure, is not really a problem: given any problem instance, we can always simulate *both* algorithms and take the best solution found, which will be good under both measures simultaneously. However, in the online setting, it is not so clear how to achieve simultaneous guarantees because we need to make the accept/reject decisions as we go.

In this paper, we describe a procedure that given algorithms A and R that are c_A-accept-competitive and c_R-reject-competitive respectively, derives a combined algorithm that is simultaneously good under both measures. Specifically, the combined algorithm is simultaneously $O(c_A)$-accept-competitive and $O(c_R c_A)$-reject-competitive.[1] The combined algorithm uses preemption — the ability to later reject a request that had previously been accepted (preempted requests are regarded as rejected) — which is known to be necessary to achieve any nontrivial guarantee in the rejection measure. At the high level, the procedure uses the following simple intuitive notion: On the one hand, if *OPT* rejects only a small fraction of the requests, the reject-competitiveness of algorithm R guarantees that it accepts a large fraction of requests and thus it is accept-competitive as well. On the other hand, if *OPT* rejects many requests, being reject-competitive is trivial (even rejecting *all* requests is fine), so algorithm A is competitive by both measures. Thus, by internally simulating both algorithms, and switching between them at just the right time, we might hope to perform well in both measures. The difficulty is showing that requests rejected by one algorithm do not adversely affect the competitive ratio of the other. Our algorithm does not use randomness to make its decisions, so the combined algorithm is deterministic unless one of the input algorithms is randomized.

In fact, we give two different combining algorithms. Both achieve $O(c_A)$-accept-competitiveness and $O(c_A c_R)$-reject-competitiveness, but the first needs to be given the value of c_A in advance, while the second does not (neither needs to know c_R). There is an added benefit to designing an algorithm that does not need to use the values of the competitive ratios; the ratios achieved by our second algorithm depend only on the behavior of A and R on the specific input sequence. For example, if algorithms A and R have log-factor competitive ratios, but happen to be constant-competitive on the actual input sequence, then the combined algorithm is constant-competitive overall (for that input sequence).

In addition to our main result, we show how to apply known techniques to combine several admission control algorithms so that the result performs nearly as well (according to either the cost or benefit measure) on any given input sequence as the best of them. More specifically, given k algorithms, we can construct a combined randomized preemptive algorithm that is $O(\log k)$-reject-competitive to the best algorithm among the k using techniques of Baeza-Yates et al. [8], Fiat et al. [11], and Azar et al. [7]. Alternately, we can construct a combined randomized preemptive algorithm that is $O(\log k)$-accept-competitive to the best algorithm among the k using techniques of Awerbuch et al. [2]. These two combined algorithms can be combined to one master algorithm using our main result to guarantee both rejection and acceptance competitiveness.

1.1 Applications of the main result

Our main result can be applied to several problems. One, which motivated this work, is admission control and call control on the line graph. Requests are intervals on the line and each edge in the graph has a *capacity*, which is the maximum number of accepted requests that can use that edge. Adler and Azar [1] give a constant accept-competitive algorithm for the problem and Blum et al. [9] give a constant

[1]This paper is based on conference papers by Azar et al. [6] and Bunde and Mansour [10]. The first of these had a worse $O(c_A^2)$ guarantee on accept-competitiveness and furthermore needed to be given c_A and c_R in advance, as well as the ability to compute OPT over the requests seen in the past. The second paper improved the accept-competitive ratio to $O(c_A)$ and removed the need to compute OPT or to know c_A or c_R.

reject-competitive algorithm for the same problem. We conclude that there is a combined algorithm that is simultaneously constant competitive for both measures. What is interesting here is that the algorithms are almost polar opposites. For example, if the capacity of each edge is 2, and three requests share an edge, the algorithm of Blum et al. [9] rejects the two "outside" requests (the one that extends farthest to the left and the one that extends farthest to the right) but the algorithm of Adler and Azar [1] rejects the one in the middle.

Our result can also be applied when the graph is a tree and accepted requests must be disjoint. Awerbuch et al. [4] and Awerbuch et al. [5] give $O(\log d)$-competitive randomized (non-preemptive) algorithms for maximizing the number of accepted requests when d is the diameter of the tree. Blum et al. [9] shows a constant competitive algorithm for the number of rejected requests in this case. By combining these two, we get an algorithm that is simultaneously $O(\log d)$-accept-competitive and $O(\log d)$-reject-competitive.

Another application is the admission control problem on general graphs where each edge is of logarithmic capacity and each request is for a fixed path. Awerbuch et al. [3] provide an $O(\log n)$-accept-competitive non-preemptive algorithm and Blum et al. [9] provide an $O(\log n)$-reject-competitive (preemptive) algorithm. We conclude there are algorithms simultaneously $O(\log n)$-accept-competitive and $O(\log^2 n)$-reject-competitive.

We should remark that for many natural online problems it is impossible to achieve competitiveness in the rejection measure and hence in both measures. For example, if the online algorithm can be forced to reject a request while the offline might have not rejected any requests, then the algorithm has an unbounded competitive ratio.

2 Model

We assume an abstract model where one request arrives at every time unit. Either the request is served (with benefit one and cost zero), or the request is rejected (with benefit zero and cost one). A request can also be preempted, in which case its benefit is set to zero and its cost is set to one. In this abstract model, the only assumption we make about the resource constraints (which are what prevent us from accepting every request) is monotonicity: if F is a feasible set of requests, then any subset of F is feasible as well. Given a sequence σ and algorithm ALG, let $\text{BENEFIT}^{ALG}(\sigma)$ be the number of requests served by ALG and $\text{COST}^{ALG}(\sigma)$ be the number of requests rejected by ALG. By definition, the sum of benefit and cost is always the number of time steps, i.e. $\text{BENEFIT}^{ALG}(\sigma)+\text{COST}^{ALG}(\sigma)=|\sigma|$ for all algorithms ALG.

An optimal algorithm OPT can either maximize the benefit $\text{BENEFIT}^{OPT}(\sigma)$ or minimize the cost $\text{COST}^{OPT}(\sigma)$. Note that for any input sequence, the optimal schedule is identical for both maximizing benefit and minimizing cost.

We are given two algorithms. The first is a possibly randomized preemptive algorithm A that guarantees a competitive ratio of $c_A \geq 1$ for benefit. That is, for any sequence σ

$$E\left[\text{BENEFIT}^{A}(\sigma)\right] \geq \frac{1}{c_A}\text{BENEFIT}^{OPT}(\sigma) \ .$$

In addition, we are given a possibly randomized preemptive algorithm R that has a guarantee of

$c_R \geq 1$ for cost. That is, for any sequence σ

$$E\left[\text{COST}^R(\sigma)\right] \leq c_R \text{COST}^{OPT}(\sigma) \ .$$

Notation: Given an input sequence σ, denote by σ_t the requests arriving by time t. As a convention, the first request is number 1.

3 Algorithm S2

Now we describe $S2$, our first combining algorithm.[2] Internally, $S2$ simulates algorithms A and R on the input σ_t. That is, $S2$ keeps track of what requests would have been accepted or rejected had it followed algorithm A from the start, and which would have been accepted or rejected had it followed algorithm R from the start. If either algorithm is randomized, then the simulation is just of a single execution (not an average over multiple runs), and our definition of quantities such as $\text{COST}^R(\sigma)$ (e.g., Lemma 3.1 and Lemma 3.2 below) are with respect to the execution observed. At any time, $S2$ is in either an A phase or an R phase. We call the algorithm corresponding to the current phase the *phase algorithm*. Algorithm $S2$ accepts, rejects, and preempts requests in exactly the same way as the phase algorithm. Algorithm $S2$ is in an R phase if $\text{COST}^R(\sigma_t)/t \leq \tau$, where $\tau = 1/(8c_A)$, and in an A phase otherwise. Whenever $S2$ switches phases, it preempts any accepted requests that the new phase algorithm did not accept. Thus, the requests accepted by $S2$ are feasible since they are a subset of the requests accepted by the phase algorithm.

3.1 Analysis of rejections

We define *requests rejected because of algorithm A* to be the requests rejected or preempted during an A phase (including those rejected when switching *to* an A phase) and denote their number at time t with $R^A(\sigma_t)$. Thus, $\text{COST}^{S2}(\sigma_t) \leq R^A(\sigma_t) + \text{COST}^R(\sigma_t)$.

Lemma 3.1. *At any time t, $R^A(\sigma_t) < \text{COST}^R(\sigma_t)/\tau$.*

Proof. If $S2$ is in an A phase at time t, $\text{COST}^R(\sigma_t) > \tau t$. Since algorithm A cannot reject more than t requests, $R^A(\sigma_t) \leq t < \text{COST}^R(\sigma_t)/\tau$.

Now consider the case that $S2$ is in an R phase at time t. Let T be the last time when $S2$ was in an A phase. By the reasoning above, $R^A(\sigma_T) < \text{COST}^R(\sigma_T)/\tau$. Since $S2$ has been in an R phase since time $T+1$, $R^A(\sigma_t) = R^A(\sigma_T)$. Also, since the number of rejections of R is a non-decreasing function of time, $\text{COST}^R(\sigma_T) \leq \text{COST}^R(\sigma_t)$. Thus, $R^A(\sigma_t) < \text{COST}^R(\sigma_t)/\tau$. □

Since $\text{COST}^{S2}(\sigma_t) \leq R^A(\sigma_t) + \text{COST}^R(\sigma_t)$, Lemma 3.1 implies that

$$\text{COST}^{S2}(\sigma_t) \leq (1 + 1/\tau)\text{COST}^R(\sigma_t) \ .$$

[2] We call the algorithm $S2$ because it is based on and improves over a previous algorithm "SWITCH" [6].

Note that if algorithm R is randomized, then the above holds for the specific execution simulated by algorithm $S2$. Now, using the fact that algorithm R is c_R-reject-competitive, we can bound the reject-competitive ratio of $S2$ by

$$\frac{E\left[\text{COST}^{S2}(\sigma_t)\right]}{\text{COST}^{OPT}(\sigma_t)} \le \frac{\left(1+\frac{1}{\tau}\right)E\left[\text{COST}^{R}(\sigma_t)\right]}{\text{COST}^{OPT}(\sigma_t)} \le \left(1+\frac{1}{\tau}\right)c_R = O(c_A c_R) \ .$$

3.2 Analysis of acceptances

We define *requests rejected because of algorithm R* to be the requests rejected or preempted during an R phase and denote their number at time t with $R^R(\sigma_t)$.

Lemma 3.2. *At any time t, $R^R(\sigma_t) \le \text{BENEFIT}^{OPT}(\sigma_t)/(7c_A)$.*

Proof. If time t is during an R phase, the lemma follows from

$$\text{BENEFIT}^{OPT}(\sigma_t) \ge \text{BENEFIT}^{R}(\sigma_t) \ge (1-\tau)t \ge 7t/8$$

and $R^R(\sigma_t) \le \text{COST}^R(\sigma_t) \le \tau t = t/(8c_A)$.

Consider time t in an A phase. If $S2$ has not had an R phase, $R^R(\sigma_t) = 0$ so the lemma holds. Otherwise, let t' be the time at which the latest R phase ended. By the argument above,

$$R^R(\sigma_{t'}) \le \text{BENEFIT}^{OPT}(\sigma_{t'})/(7c_A) \ .$$

Since $S2$ was in an A phase since time t',

$$R^R(\sigma_t) = R^R(\sigma_{t'}) \le \text{BENEFIT}^{OPT}(\sigma_{t'})/(7c_A) \ .$$

Since optimal benefit is non-decreasing with the input length, $R^R(\sigma_t) \le \text{BENEFIT}^{OPT}(\sigma_t)/(7c_A)$. □

Now we can prove that $S2$ is $O(c_A)$-accept-competitive. We do this by bounding the number of requests accepted by both algorithms, which is a lower bound on the number of requests accepted by $S2$. Since algorithm A is c_A-accept-competitive, $E\left[\text{BENEFIT}^A(\sigma)\right] \ge \text{BENEFIT}^{OPT}(\sigma)/c_A$. By Lemma 3.2, algorithm R causes $R^R(\sigma_t) \le \text{BENEFIT}^{OPT}(\sigma)/(7c_A)$ additional rejections. Thus,

$$E\left[\text{BENEFIT}^{S2}(\sigma)\right] \ge \text{BENEFIT}^{OPT}(\sigma)/c_A - \text{BENEFIT}^{OPT}(\sigma)/(7c_A) = (6/(7c_A))\text{BENEFIT}^{OPT}(\sigma)$$

and the accept-competitive ratio of $S2$ is at most $(7/6)c_A = O(c_A)$.

4 Algorithm RO

Now we define *RO*, our second combining algorithm.[3] One problem with our previous algorithm $S2$ is that, while simple, it required knowing c_A in advance. Algorithm *RO* is a bit more complicated but

[3]We call this algorithm *RO* for Ratio Oblivious because it does not need to know the competitive ratios of the input algorithms.

does not need to be given either of the competitive ratios as input. Internally, algorithm *RO* keeps times t_A and t_R, plus input prefixes σ_A and σ_R of these lengths. It maintains simulations of algorithms A and R on inputs σ_A and σ_R respectively. Whenever either of these algorithms decides to reject a request, that request is marked. Times t_A and t_R advance in phases, pausing and resuming the simulations as necessary so that $\max\{t_A, t_R\} = t$ at time t. Specifically, phase $k \geq 0$ has an R subphase, during which time t_R advances until $\text{COST}^R(\sigma_R) = 4^k$, followed by an A subphase, during which time t_A advances until $\text{BENEFIT}^A(\sigma_A) = 8 \cdot 4^k$ (note that a subphase may correspond to an empty set of requests). When a new request arrives, *RO* accepts it if the resulting set of accepted requests is feasible. While the resulting set is not feasible, *RO* preempts an arbitrary marked request (some such request must exist since $\max\{t_A, t_R\} = t$). The idea of using marks to delay rejections as long as possible is called *lazy rejection*.

4.1 Analysis of rejections

To analyze rejections, we first show that the algorithm maintains the invariant that $\text{BENEFIT}^A(\sigma_A) \leq 32\text{COST}^R(\sigma_R)$. (If either A or R is randomized, then this statement is with respect to the specific execution of each algorithm performed by *RO*.) During the first R subphase, the inequality holds vacuously because $\text{BENEFIT}^A(\sigma_A) = 0$. During the first A subphase, $\text{COST}^R(\sigma_R) = 1$ and $\text{BENEFIT}^A(\sigma_A) \leq 8$. Finally, during phases after the first, $\text{COST}^R(\sigma_R) \geq 4^{k-1}$ and $\text{BENEFIT}^A(\sigma_A) \leq 8 \cdot 4^k$.

Using the above inequality, we can bound $\text{COST}^A(\sigma_A)$ from above by

$$\begin{aligned} \text{COST}^A(\sigma_A) \leq t_A &= \text{COST}^{OPT}(\sigma_A) + \text{BENEFIT}^{OPT}(\sigma_A) \\ &\leq \text{COST}^{OPT}(\sigma_A) + c_A E\left[\text{BENEFIT}^A(\sigma_A)\right] \\ &\leq \text{COST}^{OPT}(\sigma_A) + 32 c_A E\left[\text{COST}^R(\sigma_R)\right] . \end{aligned}$$

Thus, the rejection competitive ratio of *RO* is at most

$$\frac{E\left[\text{COST}^A(\sigma_A) + \text{COST}^R(\sigma_R)\right]}{\text{COST}^{OPT}(\sigma_t)} \leq 1 + \frac{32 c_A E\left[\text{COST}^R(\sigma_R)\right]}{\text{COST}^{OPT}(\sigma_t)} + c_R = O(c_A c_R) .$$

4.2 Analysis of acceptances

To show that algorithm *RO* is $O(c_A)$-accept-competitive, we show

$$\text{BENEFIT}^{RO}(\sigma_t) \geq (1/2)\text{BENEFIT}^A(\sigma_t)$$

for all times t, all inputs, and all sequences of random bits. The desired result then follows from the competitiveness of algorithm A.

Since *RO* does not reject any requests during the first R subphase, it is optimal and $\text{BENEFIT}^{RO}(\sigma_t) = \text{BENEFIT}^A(\sigma_t)$. The first A subphase begins when algorithm R rejects (or preempts) a request C. Algorithm *RO* accepts the same requests as algorithm A except possibly for C. However, $\text{BENEFIT}^{RO}(\sigma_t) \geq 1$ since *RO* uses lazy rejection. (This is the only part of the argument in which we use lazy rejection, but this property is necessary since otherwise *RO* may reject every request when it begins the first

A subphase.) Thus, $\text{BENEFIT}^{RO}(\sigma_t) \geq (1/2)\text{BENEFIT}^{A}(\sigma_t)$. After the first subphase, $\text{COST}^{R}(\sigma_R) \leq (1/2)\text{BENEFIT}^{A}(\sigma_A)$ since $\text{COST}^{R}(\sigma_R) \leq 4^k$ and $\text{BENEFIT}^{A}(\sigma_A) \geq 8 \cdot 4^{k-1} = 2 \cdot 4^k$. Using this, we get

$$\begin{aligned} \text{BENEFIT}^{RO}(\sigma_t) &\geq t - \text{COST}^{A}(\sigma_A) - \text{COST}^{R}(\sigma_R) \\ &\geq (t - t_A) + \text{BENEFIT}^{A}(\sigma_A) - \text{COST}^{R}(\sigma_R) \\ &\geq (t - t_A) + (1/2)\text{BENEFIT}^{A}(\sigma_A) \ . \end{aligned}$$

Since BENEFIT^{A} cannot increase faster than new requests arrive, $\text{BENEFIT}^{RO}(\sigma_t) \geq (1/2)\text{BENEFIT}^{A}(\sigma_t)$.

5 Combining admission control algorithms

In this section we briefly describe how to combine a collection of online algorithms into one master algorithm that performs on any input sequence nearly as well as the best algorithm from the collection. This is done separately for the acceptance problem and for the rejection problem. Results of this form already exist in the literature [2, 7, 8, 11] but our main point here is that (a) these known techniques can be applied in our abstract model, and (b) using our main result we can combine the two master algorithms that result into one combined algorithm that guarantees both rejection and acceptance competitiveness.

The main ingredient in the combining algorithms is the process for switching between algorithms. Note that switching algorithms might mean that we need to preempt some or all requests that we currently serve. In fact, the combining algorithms have a very different structure, depending on whether they are minimizing the number of rejected requests or maximizing the number of accepted requests. The algorithms to be combined can be either randomized or deterministic.

5.1 Combining algorithms to minimize rejection

First we show how to combine k (possibly preemptive) online algorithms $R_1, R_2, \ldots, R_k$ into a master algorithm that for any sequence is reject-competitive with the best algorithm among the k for the given sequence. For a sequence of requests σ let $\text{COST}^*(\sigma) = \min_i \text{COST}^{R_i}(\sigma)$. We construct a deterministic preemptive online combining algorithm REJ_{det} such that for any σ, we have $\text{COST}^{REJ_{det}}(\sigma) = O(k\text{COST}^*(\sigma))$. We also provide a randomized preemptive online algorithm REJ_{rand} that guarantees

$$E\left[\text{COST}^{REJ_{rand}}(\sigma)\right] = O(\text{COST}^*(\sigma)\log k) \ .$$

The deterministic algorithm REJ_{det} uses a simple greedy strategy. Let $\min(t) = min\{\text{COST}^{R_i}(\sigma_t)\}$. The algorithm REJ_{det} at time t follows one of the algorithms that made fewest rejections, i.e. $min(t)$. It preempts all the requests that the selected algorithm rejected or preempted. In the worst case REJ_{det} might reject $k \cdot min(t)$ requests up to time t, establishing the following theorem.

Theorem 5.1. *The deterministic algorithm REJ_{det} rejects at most $k\text{COST}^*(\sigma)$ for any sequence σ of requests.*

The randomized algorithm REJ_{rand} uses a simple doubling strategy. Initially, it accepts all requests as long as possible with no rejection and then sets $\lambda = 1$. When it is not possible to avoid a rejection, it chooses a random i such that $\text{COST}^{R_i}(\sigma) \leq \lambda$. Whenever the condition $\text{COST}^{R_i}(\sigma) \leq \lambda$ is violated, it sets $\lambda \leftarrow 2\lambda$ and chooses a random i such that $R_i(\sigma) \le \lambda$. (If such a value of i does not exist then the condition is immediately violated and we double the value of λ.)

To bound the performance of this algorithm, we observe that our problem can be reduced to the problem of *layered graph traversal* of disjoint paths tied together at a common source. The input for layered graph traversal is a weighted graph whose vertices are partitioned into sets $L_0, L_1, \ldots, L_k$ and whose edges connect vertices in sets with adjacent indices. The objective is to move from a specified source vertex in L_0 to a specified target vertex in L_k. Initially, neither the target vertex nor the edge weights are known to the algorithm. The weights of edges between L_i and L_{i+1} become known when the algorithm visits a vertex in L_i. The target vertex becomes known when the algorithm visits a vertex in L_{k-1}. In the disjoint path version of the problem, the graph is the union of paths that are disjoint except for the source vertex.

Our problem can be reduced to disjoint paths layered graph traversal on a graph where each path corresponds to an algorithm and the i^{th} edge on a path has weight equal to the number of requests rejected when the i^{th} request arrives. (Note that this cost can be more than one if the algorithms being combined are preemptive since the arrival of a request may cause some previously-accepted requests to be preempted.) Actually, the reduction is to a slight variation of the problem since we are allowed to return to the common source with no cost and may only need to pay part of the cost when we switch to another path. This can only reduce the competitive ratio since it may help the online algorithm but it does not help the optimum (since the optimum does not switch between paths). Applying the results of Fiat et al. [11] and Azar et al. [7] to our problem gives the following theorem:

Theorem 5.2. *The expected number of requests rejected by randomized algorithm REJ_{rand} is at most $O(\log k)$ times $\text{COST}^*(\sigma)$ for any sequence of requests σ.*

Clearly, we can apply the above theorems to a case where we have k algorithms and for each input sequence σ there exists i such that $E\left[\text{COST}^{R_i}(\sigma)\right] \le c_R \text{COST}^{OPT}(\sigma)$:

Corollary 5.3. *When constructed from k algorithms such that for any input sequence at least one algorithm is c_R-reject-competitive for that sequence, the deterministic algorithm REJ_{det} is $O(c_R k)$-reject-competitive and the randomized algorithm REJ_{rand} is $O(c_R \log k)$-reject-competitive.*

5.2 Combining algorithms to maximize acceptance

Now we show how to combine k non-preemptive algorithms $A_1, A_2, \ldots, A_k$ into a master algorithm that is accept-competitive with the best algorithm among the k for the given sequence. For a sequence of requests σ let $A^*(\sigma) = \max_i \text{BENEFIT}^{A_i}(\sigma)$. We construct one randomized preemptive online algorithm *ACC* such that for any σ we have

$$E\left[\text{BENEFIT}^{ACC}(\sigma)\right] \ge A^*(\sigma)/\log k \;.$$

As before, we combine the algorithms by switching between them. When switching to a certain algorithm we might need to preempt all requests we currently have, and in the worst case we are left with a single accepted request. This suggests that there is no deterministic competitive combining algorithm so we use randomization in our combining algorithm.

The basic idea is that our generic model is a variant of the problem of picking a winner [2]. In the problem of picking a winner we have k options (algorithms, in our setting). At any time some options yield a benefit of one, while the others have a benefit of zero. (Negative benefits would be possible if the algorithms being combined are preemptive, which is why we restrict our attention to non-preemptive algorithms.) The decision maker (our combining algorithm) switches between options. When switching, the decision maker loses all its current benefit and gets, from that time on, the benefit of the current option. Switching between options corresponds in our setting to switching between algorithms while possibly preempting all currently accepted requests. It is shown by Awerbuch et al. [2] that using polylogarithmic number of switches, the decision maker achieves benefit at least a $O(\log k)$ fraction of the benefit yielded by the best choice with high probability. Therefore,

Theorem 5.4. *The expected number of requests accepted by the randomized algorithm ACC is at least a $O(\log k)$ fraction of $A^*(\sigma)$ for any sequence of requests σ.*

As before, we can apply the above theorem to the case where we have k algorithms and for each input sequence σ there exists i such that $A_i(\sigma) \geq \mathrm{OPT}(\sigma)/c_A$:

Corollary 5.5. *When constructed from k algorithms such that for any input sequence at least one algorithm is c_A-accept-competitive for that sequence, the algorithm ACC is $O(c_A \log k)$-accept-competitive.*

In fact, by slightly modifying the algorithm of Awerbuch et al. [2], allowing the combining algorithm to disengage from all options, one can extend these results to the case of preemptive algorithms.

6 Conclusions and open problems

We have described procedures that take an algorithm A with competitive ratio c_A for benefit, and an algorithm R with competitive ratio c_R for cost, produce an online algorithm that simultaneously achieves competitive ratio $O(c_A)$ for benefit and $O(c_A c_R)$ for cost. We do not know if it is possible in general to do better. In particular, an ideal result in this direction would achieve $O(c_A)$ for benefit and $O(c_R)$ for cost simultaneously.

Acknowledgements. The authors would like to thank the referees for their helpful comments. David would like to acknowledge Sariel Har-Peled for help performing the analysis of $S2$ and RO when the input algorithms are randomized.

References

[1] * R. ADLER AND Y. AZAR: Beating the logarithmic lower bound: randomized preemptive disjoint paths and call control algorithms. *J. of Scheduling*, pp. 113–129, 2003. Prelimary version in Proc. 10th ACM-SIAM Symp. on Discrete Algorithms, pp. 1–10, 1999. [doi:10.1023/A:1022933824889, SODA:314500.314512, JSched:jq27641706456655]. 1.1

[2] * B. AWERBUCH, Y. AZAR, A. FIAT, AND T. LEIGHTON: Making commitments in the face of uncertainty: How to pick a winner almost every time. In *Proc. 28th ACM Symp. on Theory of Computing*, pp. 519–530, 1996. [STOC:237814.238000]. 1, 5, 5.2, 5.2

[3] * B. AWERBUCH, Y. AZAR, AND S. PLOTKIN: Throughput-competitive online routing. In *Proc. 34th IEEE Symp. on Foundations of Computer Science*, pp. 32–40, 1993. [FOCS:10.1109/SFCS.1993.366884]. 1.1

[4] * B. AWERBUCH, Y. BARTAL, A. FIAT, AND A. ROSÉN: Competitive non-preemptive call control. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pp. 312–320, 1994. [SODA:314464.314510]. 1.1

[5] * B. AWERBUCH, R. GAWLICK, T. LEIGHTON, AND Y. RABANI: On-line admission control and circuit routing for high performance computation and communication. In *Proc. 35th IEEE Symp. on Foundations of Computer Science*, pp. 412–423, 1994. [FOCS:10.1109/SFCS.1994.365675]. 1.1

[6] * Y. AZAR, A. BLUM, AND Y. MANSOUR: Combining online algorithms for rejection and acceptance. In *Proc. 15th ACM Symp. Parallelism in Algorithms and Architectures*, pp. 159–163, 2003. [SPAA:10.1145/777412.777438]. 1, 2

[7] * Y. AZAR, A. BRODER, AND M. MANASSE: On-line choice of on-line algorithms. In *Proc. 4th ACM-SIAM Symp. on Discrete Algorithms*, pp. 432–440, 1993. [SODA:313559.313847]. 1, 5, 5.1

[8] * R. BAEZA-YATES, J. CULBERSON, AND G. RAWLINS: Searching in the plane. *Information and Computation*, 106(2):234–252, 1993. Preliminary version in Proc. 1st Scandinavian Workshop on Algorithm Theory, LNCS 318, pp. 176–189, 1988. [IandC:10.1006/inco.1993.1054]. 1, 5

[9] * A. BLUM, A. KALAI, AND J. KLEINBERG: Admission control to minimize rejections. *Internet Mathematics*, 1(2):165–176, 2004. Preliminary version in Proc. 7th Workshop on Algorithms and Data Structures, LNCS 2125, pp. 155–164, 2001. [WADS:9ukehq56d6yp2m22, InternetMath:1(2):165–176]. 1.1

[10] * D.P. BUNDE AND Y. MANSOUR: Improved combination of online algorithms for acceptance and rejection. In *Proc. 16th ACM Symp. Parallelism in Algorithms and Architectures*, pp. 265–266, 2004. [SPAA:10.1145/1007912.1007952]. 1

[11] * A. FIAT, D. FOSTER, H. KARLOFF, Y. RABANI, Y. RAVID, AND S. VISHWANATHAN: Competitive algorithms for layered graph traversal. *SIAM J. on Computing*, 28(2):447–462, 1998. Preliminary version in Proc. 32nd IEEE Symp. on Foundations of Computer Science, pp 288–297, 1991. [FOCS:10.1109/SFCS.1991.185381, SICOMP:10.1137/S0097539795279943]. 1, 5, 5.1

AUTHORS

Yossi Azar
professor
School of Computer Science
Tel-Aviv University
Tel-Aviv, 69978, Israel
azar@cs.tau.ac.il
http://www.cs.tau.ac.il/~azar/

Avrim Blum
professor
Department of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213-3891
avrim@cs.cmu.edu
http://www.cs.cmu.edu/~avrim/

David P. Bunde
graduate student
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
bunde@uiuc.edu
http://compgeom.cs.uiuc.edu/~bunde/

Yishay Mansour
professor
School of Computer Science
Tel-Aviv University
Tel-Aviv, 69978, Israel
mansour@cs.tau.ac.il
http://www.math.tau.ac.il/~mansour/

ABOUT THE AUTHORS

YOSSI AZAR received his Ph.D. from Tel-Aviv University in 1989 (supervised by Noga Alon). He spent several years in the Bay Area (Stanford, DEC, IBM); his experience there included the Loma Prieta earthquake, 7.1 on the Richter scale. In 1994 he joined the computer science faculty at Tel-Aviv University. He was the chair of the department between 2002 and 2004. His main research interests are in the theory of algorithms, especially online, randomized and approximation algorithms, as well as in trying to understand his three children.

AVRIM BLUM grew up in Berkeley, CA, and then went to MIT for undergraduate and graduate school. He received his Ph.D. in Computer Science under the supervision of Ron Rivest, and now works in the family business. His research interests include approximation algorithms, online algorithms, and machine learning theory. He has two children, Alex and Aaron, who may or may not go into the family business.

DAVID BUNDE is currently pursuing his Ph.D. in the Computer Science department at the University of Illinois in Urbana-Champaign, supervised by Jeff Erickson. Most of his research has been on scheduling and processor allocation, though he also likes to work on other algorithmic problems like the current paper and graph pebbling. In his spare time, he enjoys reading and playing strategy games, particularly Civilization III.

YISHAY MANSOUR obtained his B.A. in 1985 and his M.Sc. in 1987 at the Technion; his M.Sc. advisor was Prof. Shmuel Zaks. He completed his Ph.D. at MIT in 1990 under the supervision of Professors Shafi Goldwasser and Baruch Awerbuch. Subsequently he became a postdoctoral fellow at Harvard University and a Research Staff Member at IBM T.J. Watson Research Center. Since 1992 he has been with the School of Computer Science at Tel-Aviv University, where he was the chairman during 2000-2002. His research interests include online algorithms, communication networks, machine learning, reinforcement learning and the theory of computation.