Improved Combination of Online Algorithms for Acceptance and Rejection

David P. Bunde*
Department of Computer Science
Univ. Illinois at Urbana-Champaign
Urbana, IL 61801
bunde@uiuc.edu

Yishay Mansour†
School of Computer Science
Tel-Aviv University
Tel-Aviv, 69978, Israel
mansour@tau.ac.il

ABSTRACT

Given two admission control algorithms that are c_A-accept-competitive and c_R-reject-competitive respectively, we give two ways to make an algorithm that is simultaneously $O(c_A)$-accept-competitive and $O(c_A c_R)$-reject-competitive. The combined algorithms make no reference to the offline optimal solution. In addition, one of the algorithms does not require knowing the value of either c_A or c_R. This improves on work of Azar, Blum, and Mansour, whose combined algorithm was $O(c_A^2)$-accept-competitive, involved computing offline optimal solutions, and required knowing the values of both c_A and c_R.

Categories and Subject Descriptors

C.2.2 [**Computer-Communication Networks**]: Network Protocols—*Routing protocols*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms

Algorithms, Theory.

Keywords

On-line, Competitive, QoS, Admission control.

1. INTRODUCTION

In *admission control* problems, the input is a list of *calls*, each a request for resources. The algorithm is given limited resources and chooses a subset of the calls to satisfy, rejecting the others. Typically, these decisions must be made online, where it is not possible to find an optimal solution. Two types of competitive algorithms have been devised. One type of algorithm tries to maximize the value of accepted calls, called its *benefit* and denoted $B^{ALG}(\sigma)$ for input σ. We say that an online algorithm A is *c-accept-competitive* if $B^A(\sigma) \geq (1/c)B^{OPT}(\sigma)$ for all σ, where OPT is the offline optimal algorithm. The other type of competitive algorithm tries to minimize the value of rejected calls, called its *cost* and denoted $C^{ALG}(\sigma)$. An online algorithm R is *c-reject-competitive* if $C^R(\sigma) \leq cC^{OPT}(\sigma)$ for all σ. Although OPT both maximizes benefit and minimizes cost, accept-competitive and reject-competitive algorithms are incomparable in general. For example, consider a 2-accept-competitive algorithm A and a 2-reject-competitive algorithm R. If 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.

Azar, Blum, and Mansour [1] cite examples of each type of algorithm and give an algorithm $SWITCH$ to combine a c_A-accept-competitive algorithm A and a c_R-reject-competitive algorithm R into an algorithm that is simultaneously $O(c_A^2)$-accept-competitive and $O(c_A c_R)$-reject-competitive. The main idea behind $SWITCH$ is to alternately run algorithms A and R depending on the proportion of calls accepted by an offline optimal solution for σ_t, the input up to time t. It requires explictly knowing the values of c_A and c_R. In this paper, we give two algorithms that are simultaneously $O(c_A)$-accept-competitive and $O(c_A c_R)$-reject competitive. Neither requires finding optimal solutions. The first, $S2$, is a modification of $SWITCH$ that no longer requires knowing the value of c_R. The second, RO, is new and does not require knowing the value of either c_A or c_R.

Model. Our model is identical to that of Azar et al. [1]. One call arrives at each time step. If the call is accepted and not later preempted, the algorithm accrues benefit one. If the call is rejected (or later preempted), it accrues cost one. Thus, $|\sigma| = B^{ALG}(\sigma) + C^{ALG}(\sigma)$ for all inputs σ and all algorithms ALG. The only assumption about the resources made by algorithm $S2$ is monotonicity; any subset of a feasible set of calls is also feasible. In addition, algorithm RO must be able to tell when two calls cannot be satisfied concurrently, but neither algorithm requires an explicit representation of the resources.

*Partially supported by NSF grant CCR 0093348.
†Partially supported by the Israel Science Foundation.

SPAA'04, June 27–30, 2004, Barcelona, Spain.
ACM 1-58113-840-7/04/0006.

2. ALGORITHM S2

Now we describe algorithm $S2$. Internally, $S2$ simulates algorithms A and R on the input. 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.* $S2$ accepts, rejects, and preempts calls in exactly the same way as the phase algorithm. At each time step, $S2$ decides its current phase by calculating $r_t = B^R(\sigma_t)/t$. $S2$ is in an R phase if $r_t \geq \tau = 1 - \bar{\tau}$, where $\bar{\tau} = 1/(8c_A)$, and in an A phase if $r_t < \tau$. Whenever $S2$ switches phases, it preempts any accepted calls that the new phase algorithm did not accept. Thus, the calls accepted by $S2$ are feasible since they are a subset of the calls accepted by the phase algorithm.

2.1 Analysis of Rejections

If $S2$ is in an A phase at time t, $C^R(\sigma_t) \geq \bar{\tau}t$. Since algorithm R is c_R-reject-competitive, $C^{OPT}(\sigma_t) \geq \bar{\tau}t/c_R = t/(8c_Ac_R)$. Even if $S2$ rejects every call, its rejection competitive ratio is at most $8c_Ac_R = O(c_Ac_R)$.

Now suppose that $S2$ began an R phase at time $T+1$ and is still in it at time $T+t$. We may assume $T > 0$ since otherwise $S2$ has been in an R phase since the beginning of the input and is thus c_R-reject-competitive. Since $S2$ was in an A phase at time T, $C^R(\sigma_T) > \bar{\tau}T$. Thus, $C^{OPT}(\sigma_T) > \bar{\tau}T/c_R = T/(8c_Ac_R)$. Adding calls cannot decrease rejections so $C^{OPT}(\sigma_{T+t}) \geq C^{OPT}(\sigma_T) \geq T/(8c_Ac_R)$. $S2$ rejected at most T calls before the current R phase and at most $C^R(\sigma_{T+t})$ during the R phase. Thus, the rejection competitive ratio is at most

$$\frac{T}{T/(8c_Ac_R)} + \frac{C^R(\sigma_{T+t})}{C^R(\sigma_{T+t})/c_R} = 8c_Ac_R + c_R = O(c_Ac_R)$$

2.2 Analysis of Acceptance

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

LEMMA 1. *At time t, $R^R(t) \leq B^{OPT}(\sigma_t)/(7c_A)$.*

PROOF. If time t is during an R phase, the lemma follows from $B^{OPT}(\sigma_t) \geq B^R(\sigma_t) \geq t\tau = t(1-\bar{\tau}) \geq 7t/8$ and $R^R(t) \leq C^R(\sigma_t) \leq \bar{\tau}t = t/(8c_A)$.

Consider time t in an A phase. If $S2$ has not had an R phase, $R^R(t) = 0$ so the lemma holds. Otherwise, let the latest R phase end at time t'. By the argument above, $R^R(t') \leq B^{OPT}(\sigma_{t'})/(7c_A)$. Since $S2$ was in an A phase since time t', $R^R(t) = R^R(t') \leq B^{OPT}(\sigma_{t'})/(7c_A)$. Since optimal benefit grows with the input, $R^R(t) \leq B^{OPT}(\sigma_t)/(7c_A)$. $\square$

Now we can prove that $S2$ is $O(c_A)$-accept-competitive. We do this by bounding the number of calls accepted by both algorithms, which is a lower bound on the number of calls accepted by $S2$. Since algorithm A is c_A-accept-competitive, $B^A(\sigma) \geq B^{OPT}(\sigma)/c_A$. By the lemma, algorithm R causes $R^R(t) \leq B^{OPT}(\sigma)/(7c_A)$ additional rejections. Thus, $B^{S2}(\sigma) \geq B^{OPT}(\sigma)/c_A - B^{OPT}(\sigma)/(7c_A) = 6B^{OPT}(\sigma)/(7c_A)$ and the accept-competitive ratio is at most $(7/6)c_A = O(c_A)$.

3. ALGORITHM RO

Now we describe algorithm RO. Internally, it keeps times t_A and t_R, plus input prefixes σ_A and σ_R of these lengths. (The external time t is $\max\{t_A, t_R\}$.) It simulates algorithms A and R on inputs σ_A and σ_R respectively, marking rejected calls. Times t_A and t_R advance in phases. Phase k has an R subphase, advancing time t_R until $C^R(\sigma_R) = 4^k$, followed by an A subphase, advancing time t_A until $B^A(\sigma_A) = 8 \cdot 4^k$. RO rejects marked calls that cannot be satisfied concurrently with the accepted calls. The idea of using marks to delay rejections as long as possible is called *lazy rejection.*

3.1 Analysis of Rejection

One call comes per unit time, so $C^A(\sigma_A) = t_A - B^A(\sigma_A)$. Similarly, $C^{OPT}(\sigma_A) \geq t_A - c_AB^A(\sigma_A)$, since $B^{OPT}(\sigma_A) \leq c_AB^A(\sigma_A)$. Combining these gives $C^A(\sigma_A) < C^{OPT}(\sigma_A) + c_AB^A(\sigma_A)$. During phase k, $C^R(\sigma_R) \leq 4^k$, $C^{OPT}(\sigma_R) \geq 4^{k-1}/c_R$, and $B^A(\sigma_A) \leq 8 \cdot 4^k$. Thus, the compative ratio is at most

$$\frac{4^k + C^{OPT}(\sigma_A) + c_AB^A(\sigma_A)}{C^{OPT}(\sigma_t)} = O(c_Ac_R)$$

3.2 Analysis of Acceptance

First, consider the case $k = 0$. In the R subphase, RO rejects no calls and is optimal. The A subphase begins when algorithm R rejects a call C. RO accepts the same calls as algorithm A except possibly for C. However, $B^{RO}(\sigma_t) \geq 1$ since RO uses lazy rejection. Thus, $B^{RO}(\sigma_t) \geq B^A(\sigma_t)$ if $B^A(\sigma_t) = 1$ and $B^{RO}(\sigma_t) \geq B^A(\sigma_t) - 1$ otherwise, so RO is $2c_A$-accept-competitive.

Now, consider $k > 0$ and $t_A \geq t_R$. Since $C^R(\sigma_R) \leq 4^k$ and $B^A(\sigma_A) \geq 8 \cdot 4^{k-1} = 2 \cdot 4^k$, $C^R(\sigma_R) \leq (1/2)B^A(\sigma_A)$. Thus, $B^{RO}(\sigma_t) \geq (1/2)B^A(\sigma_t)$ and RO is $2c_A$-accept-competitive.

Finally, consider $k > 0$ and $t_A < t_R$. Since $B^A(\sigma_A) \geq 8 \cdot 4^{k-1}$ and $C^R(\sigma_A) \leq C^R(\sigma_R) \leq 4^k$, $B^A(\sigma_A) \geq C^R(\sigma_A)$. Thus, $C^A(\sigma_A) = t_A - B^A(\sigma_A) \leq t_A - C^R(\sigma_A) = B^R(\sigma_A) \leq B^{OPT}(\sigma_A)$. Algorithm A is c_A-accept-competitive, so $B^{OPT}(\sigma_A) \leq c_AB^A(\sigma_A) \leq c_A 8 \cdot 4^k$, implying $C^A(\sigma_A) < c_A 8 \cdot 4^k$. With $C^R(\sigma_t) \leq 4^k$, this gives $B^{OPT}(\sigma_t) \leq t \leq B^R(\sigma_t) + 4^k \leq B^{RO}(\sigma_t) + C^A(\sigma_A) + 4^k < B^{RO}(\sigma_t) + c_A 8 \cdot 4^k + 4^k$. Because $B^A(\sigma_A) \geq 8 \cdot 4^{k-1}$ while $C^R(\sigma_R) \leq 4^k$, $B^{RO}(\sigma_t) \geq B^A(\sigma_A) - C^R(\sigma_R) \geq 4^k$. Thus, the accept competitive ratio of RO is

$$\frac{B^{OPT}(\sigma)}{B^{RO}(\sigma)} \leq 1 + \frac{c_A 8 \cdot 4^k + 4^k}{4^k} = 2 + 8c_A = O(c_A)$$

4. REFERENCES

[1] Y. Azar, A. Blum, and Y. Mansour. Combining online algorithms for rejection and acceptance. In *Proc. 15th Ann. ACM Symp. Parallelism in Algorithms and Architectures*, pages 159–163, 2003.