Combining Online Algorithms for Rejection and Acceptance

Yossi Azar[*] Avrim Blum[†] Yishay Mansour [‡]

May 28, 2003

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 calls accepted) and under cost models (e.g., with a goal of approximately minimizing the number of calls rejected). Unfortunately, algorithms designed for these two measures can often be quite different, even polar opposites (e.g., [1, 8]). In this work we consider the problem of combining algorithms designed for each of these objectives in a way that simultaneously is good under both measures. More formally, we are given an algorithm A which is c_A competitive w.r.t. the number of accepted calls and an algorithm R which is c_R competitive w.r.t. the number of rejected calls. We derive a combined algorithm whose competitive ratio is $O(c_R c_A)$ for rejection and $O(c_A^2)$ for acceptance. We also show building on known techniques that given a collection of k algorithms, we can construct one master algorithm which performs similar to the best algorithm among the k for the acceptance problem and another master algorithm which performs similar 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 which guarantees both rejection and acceptance competitiveness.

1 Introduction

Resource allocation is one of the most critical tasks in communication networks. Many resources are in constant "short supply": this includes the bandwidth (of the various links), queuing delays (or rather the lack of queuing delays in the switches), uneven scheduling (or rather 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 in certain cases requests may have to be rejected due to the lack of resources. A simple obvious example is bandwidth allocation. Suppose we have a certain link with a given capacity, and different calls have requests for bandwidth on that link. Since the system cannot allocate more then 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 will have to decide for each request whether or not to accept (and grant the request the resources) while having minimal (or no) knowledge of future requests. This leads very naturally

[*]School of Computer Science, Tel-Aviv University, Tel Aviv University, Tel-Aviv, 69978, Israel. E-Mail: azar@tau.ac.il . Research supported in part by the Israel Science Foundation and by the IST Program of the EU.

[†]School of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213-3891. E-Mail: avrim@cs.cmu.eduResearch supported in part by NSF grants CCR-0105488, ITR CCR-0122581, and ITR IIS-0121678.

[‡]School of Computer Science, Tel-Aviv University , Tel Aviv University, Tel-Aviv, 69978, Israel. E-Mail: mansour@tau.ac.il . Research supported in part by the Israel Science Foundation

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 setting, including, call control, admission control, active queue management, and switch throughput.

When one applies competitive analysis, one needs to decide what complexity measure to focus on. One can try either to minimize the number of rejected requests or alternatively to maximize the number of accepted requests. Even though the optimal solution 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 two complementary problems. There might be one algorithm A that achieves a good ratio for maximizing benefit but has a lousy ratio in terms of cost, and a totally different algorithm R that has a good ratio for minimizing cost but a lousy ratio in terms of benefit.

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

Main result: Our main result is to design an on-line algorithm that can fruitfully combine algorithms with guarantees in each of these measures to produce a single algorithm that simultaneously has a good guarantee for both. Specifically, given an algorithm A that has a competitive ratio of c_A for the number of accepted requests, and an algorithm R that has a competitive ratio of c_R for the number rejected requests, we derive a combined algorithm whose competitive ratio is $O(c_R c_A)$ for rejection, and $O(c_A^2)$ for acceptance. We should note that the combined algorithm uses preemption, i.e., the ability to preempt a request that has been accepted (preempted requests are regarded as though they have been rejected). We also note that our algorithm is deterministic if both A and R are deterministic. If A or R are randomized then obviously our algorithm is also randomized. To demonstrate the strength of our result, if both c_A and c_R are constant competitive then the combined algorithm is constant competitive.

Our combining algorithm follows a simple intuitive notion. If only a small fraction of the requests need to be rejected (small compared to $1/c_R$), we should use R. Since R is c_R-competitive for rejections, it will be accepting a large fraction of requests and thus be competitive in terms of accepts as well. On the other hand, if the optimal number of rejections becomes large, then we no longer need to worry about our rejection ratio (even rejecting *all* requests would be fine), so we should switch to algorithm A. The delicate part of our algorithm is in analyzing the switching between the two algorithms, and considering the cost of switching from one algorithm to the other in the overall analysis.

Applications of main result: Our main result can be applied to several problems. One is admission control and call control on the line. In particular, there exist constant competitive algorithms for the problem where the goal is to maximize the number of accepted requests [1] as well as constant competitive algorithms for same problem where the measure is the number of the rejected requests [8]. We conclude that there is a combined algorithm that simultaneously is constant competitive algorithm for both complementary measures. What is interesting here is that the algorithms of [1] and [8] are almost polar opposites. For example, if the capacity is 2 and there is a collision of 3 intervals, the algorithm of [8] 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 [1] rejects the one in the middle. Next, we consider call control (disjoint paths) on a tree. The papers [4, 5] show $O(\log d)$ competitive randomized (non-preemptive) algorithms for maximizing the number of accepted requests (d is the diameter of the tree). The paper [8] shows a constant competitive algorithm for the number of rejected requests. We conclude that there is one algorithm which is $O(\log^2 d)$-competitive for the number of accepted requests and $O(\log d)$-competitive for the number of rejected requests. Another application is the admission control problem on general graphs where each edge is of logarithmic capacity and each requests is for a unit demand on a fixed path. The paper [3] provides an $O(\log n)$ competitive (non-preemptive) algorithm for the number of accepted requests. The paper [8] provides an $O(\log n)$ competitive (preemptive) algorithm for the number of rejected requests. We conclude there is an $O(\log^2 n)$-competitive algorithm for both. (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 we have an unbounded competitive ratio.)

Additional results: We also consider the case that there are several (say k) online (possibly preemptive) algorithms for the admission control problem. Our goal is to provide an algorithm that performs similar to the best algorithm on any given input sequence. We consider the problem both for the measure of the number of rejected requests and for the measure of the number accepted requests. We observe that using known techniques [7, 9, 6] we can construct a combined randomized preemptive algorithm which is at most $O(\log k)$ worse with respect to the number of rejected requests of the best algorithm among the k. Using known results [2] we can also construct a combined randomized preemptive algorithm which is at most $O(\log k)$ worse with respect to the number of accepted requests of the best algorithm among the k. These two combined algorithms can be combined to one master algorithm using our main result to guarantee both rejection and acceptance competitiveness.

2 Model

We assume an abstract model where at every time unit a request is received. Either the request is served (with benefit one and cost zero), or the request is rejected (with benefit zero and cost 1). 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 too. Given a sequence σ, let $V_B(\sigma)$ be the number of requests served and $V_C(\sigma)$ the number of requests rejected. By definition, the sum of benefit and cost is always the number of time steps, i.e. $V_B(\sigma) + V_C(\sigma) = |\sigma|$.

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

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

$$E(V_B^A(\sigma)) \geq \frac{1}{c_A} V_B^{OPT}(\sigma).$$

In addition we are given a possibly randomized preemptive algorithm R that has a guarantee of $c_R \geq 1$ for the cost, namely, for any sequence σ

$$E(V_C^R(\sigma)) \leq c_R V_C^{OPT}(\sigma).$$

Notation: Given an input sequence σ, denote by $\sigma_{(T+1,T+t)}$ the sequence of requests from time $T+1$ until time $T+t$. We also write σ_t for $\sigma_{(1,t)}$. As a convention, the first request is number

1. Given a subset F of requests from σ we denote by σ_F the sub-sequence that includes only the requests of F. Given two sub-sequences σ_1 and σ_2 we denote by $\sigma_1\sigma_2$ the combined sequence of requests, which first has the requests of σ_1 followed by the requests of σ_2.

3 Our Algorithm

We assume we are given a deterministic, possibly preemptive, algorithm A which is c_A competitive for the benefit, and a deterministic, possibly preemptive, algorithm R which is c_R competitive for the cost. Our algorithm $SWITCH$ receives the two algorithms as input. At each time step, we compute the average optimal benefit so far and denote $o_t = V_B^{OPT}(\sigma_t)/t$ and $\bar{o}_t = 1 - o_t$. Our deterministic algorithm $SWITCH$ has two thresholds, u and l, where $u = 1 - \bar{u}$ and $l = 1 - 2\bar{u}$. We choose $\bar{u} = 1/(8c_Ac_R)$. Note that $u > l$.

Algorithm $SWITCH$ switches between an R phase and an A phase. During an R phase, $SWITCH$ runs the R algorithm and in an A phase it runs the A algorithm. When $SWITCH$ is in an R phase and o_t drops below l it switches to an A phase. When $SWITCH$ is in an A phase and o_t is higher than u it switches to an R phase. What remains is to describe how to initialize the algorithms when we start a phase. This will be done as follows. Let F be the set of calls served so far by $SWITCH$ at the time of the switch. We initialize the new algorithm (A or R) by feeding it the sequence σ_F which is the sequence F in the original order of σ_t. The algorithm R will accept all the requests (since OPT can serve all of them). On the other hand the algorithm A might reject or even preempt some of them, in such a case we preempt those requests. (This is where our assumption on preemption is essential.)

Theorem 1 *The deterministic preemptive algorithm $SWITCH$ is simultaneously $O(c_Rc_A)$ competitive for cost and $O(c_A^2)$ competitive for benefit, given a deterministic possibly preemptive algorithm A which is c_A competitive for benefit, and a deterministic possibly preemptive algorithm R which is c_R competitive for cost.*

We can also construct a combined algorithm if A and R are randomized. The randomized algorithm $SWITCH$ is constructed in the same way. Note that the decisions to switch between R phases and A phases are done according to the optimal value and hence are deterministic. When we start a phase, we start the algorithm (A or R) with new random bits, which are independent of any previous chosen random bits, and feed it with the actual F (Note that F is a random variable).

Theorem 2 *The randomized preemptive algorithm $SWITCH$ is simultaneously $O(c_Rc_A)$ competitive for cost and $O(c_A^2)$ competitive for benefit, given a possibly randomized preemptive algorithm A which is c_A competitive for benefit, and a possibly randomized preemptive algorithm R which is c_R competitive for cost.*

In the analysis below we assume that A and R are deterministic and hence $SWITCH$ is deterministic. The analysis for the randomized case is very similar but omitted. We just need to replace a variable by its expectation in the analysis and use the fact that the expectation of a product of *independent* random variables is the product of their expectations.

4 Analysis of Rejections

This is the simpler case. Assume that at time t our algorithm is in an A phase. This implies that the optimal schedule accepts at most ut, and therefore rejects at least $\bar{u}t$. Even if the online algorithm rejects all the requests it will still be $1/\bar{u} = 8c_Ac_R$ rejection competitive.

Assume that at time $T+1$ we started an R phase, and consider a time $T+t$, in which we are in the same R phase. Until time T the optimal algorithm rejected at least $T\bar{u}$. Assume that on the sequence of $\sigma_F\sigma_{(T+1,T+t)}$ the optimal schedule for that sequence rejects γt requests. Since algorithm R is c_R competitive we know that it will reject at most $(\gamma t)c_R$ requests. Hence, for the entire sequence $\sigma_{(T+t)}$ our algorithm rejected at most $T+\gamma t c_R$ requests. Clearly the optimal algorithm rejected more requests for the entire sequence $\sigma_{(T+t)}$ than on the sub-sequence $\sigma_F\sigma_{(T+1,T+t)}$. Hence the rejection competitive ratio is at most

$$\frac{T+\gamma t c_R}{\max\{T\bar{u},\gamma t\}} \le 1/\bar{u}+c_R = 8c_Ac_R+c_R = O(c_Ac_R)$$

5 Analysis of Acceptance

This is a somewhat tricky case. Not only will we analyze separately the two phases, but during an A phase we will separate the competitive ratio at the beginning and at the end of the phase. The last point turns out to be crucial to get any bound on the competitive ratio. Specifically, in addition to our main claim that the acceptance competitive ratio is $O(c_A^2)$ we also assume (and prove) the following two stronger invariants. The first is that during an R phase the acceptance competitive ratio is at most $a_r = 8c_A$. The second is that at the end of an A phase the competitive ratio is at most $a_e = 2c_A$. The proof below of the two invariants and the main claim is done by induction.

5.1 Switching from R to A

Assume that at time $T+1$ we switch to an A phase. We know that on the entire past the optimal schedule has a benefit of lT. Due to our assumption (first invariant) we know that $SWITCH$ has benefit of at least lT/a_r. $SWITCH$ now initializes algorithm A on this sequence σ_F, which may now reject additional requests but clearly must accept at least $(lT/a_r)/c_A$. Therefore, immediately after the switch (before handling request $T+1$) the competitive ratio is at most $a_rc_A = O(c_A^2)$.

Next we consider a time $T+t$ in the same A phase. Clearly OPT for the sequence σ_F has benefit exactly $|F|$ since it can accept all requests (as the online algorithm has them). Assume that OPT on the sequence $\sigma_F\sigma_{(T+1,T+t)}$ accepts $|F|+\beta t$ requests for some $\beta > 0$. Clearly, on that sequence $SWITCH$ has a value of at least $(|F|+\beta t)/c_A$. Moreover, by monotonicity and sub-additivity of OPT we have

$$\begin{aligned} V_B^{OPT}(\sigma_{(T+t)}) &\le V_B^{OPT}(\sigma_T)+V_B^{OPT}(\sigma_{(T+1,T+t)}) \\ &\le V_B^{OPT}(\sigma_T)+V_B^{OPT}(\sigma_F\sigma_{(T+1,T+t)}) \ . \end{aligned}$$

This implies that at time $T+t$ the acceptance competitive ratio is at most

$$\begin{aligned} \frac{lT+(|F|+\beta t)}{(|F|+\beta t)/c_A} &= c_A\frac{lT}{|F|+\beta t}+c_A \\ &\le c_A\frac{lT}{|F|}+c_A \\ &\le c_A\frac{lT}{lT/a_r}+c_A \\ &= c_Aa_r+c_A = O(c_A^2) \end{aligned}$$

as needed for our main claim.

We claim (second invariant) that at the end of the phase, at time $T+t$, the acceptance competitive ratio is at most a_e . Even if we assume that in the time interval $[T+1, T+t]$ the optimal algorithm receives all the requests, then for $t < T$ we have $lT + t < u(t+T)$, for our choice of u and l. Therefore, the duration of an A phase which starts at time T is at least T,i.e. $t \geq T$. Also

$$\begin{aligned}
V_B^{OPT}(\sigma_F \sigma_{(T+1,T+t)}) &\geq V_B^{OPT}(\sigma_{(T+1,T+t)}) \\
&\geq V_B^{OPT}(\sigma_{(T+t)}) - V_B^{OPT}(\sigma_T) \\
&= u(t+T) - lT \\
&> ut \ .
\end{aligned}$$

Hence the value of $SWITCH$ at time $T+t$ is at least uT/c_A and therefore the competitive ratio is at most

$$\frac{u(T+t)}{ut/c_A} = c_A \frac{T+t}{t} \leq 2c_A = a_e$$

as needed for the second invariant.

5.2 Switching from A to R

Assume that at time $T+1$ we switch from an A phase to an R phase. This implies that on the input σ_T the benefit of the optimal offline, OPT, is uT. From our assumptions (second invariant) we have that $SWITCH$ has benefit at least $uT/a_e = uT/(2c_A)$. At the time of the switch we are re-inputting the requests in the online memory to the algorithm R. Since OPT can accept all the request we know that R will accept all of them. Hence the competitive ratio remains a_e (before handling request $T+1$).

Consider the time $T+t$ in the same R phase. By definition OPT accepted on the sequence $\sigma_{(T+t)}$ exactly $(T+t)o_{(T+t)}$ requests and rejected $(T+t)\bar{o}_{(T+t)}$ requests. Hence OPT rejected at most that number on the sequence $\sigma_F \sigma_{(T+1,T+t)}$ since it is a sub-sequence. Therefore R rejected at most $c_R(T+t)\bar{o}_{(T+t)}$ requests on that sequence. Hence the number of accepted requests at time $T+t$ is at least

$$\begin{aligned}
|F| &+ t - c_R(T+t)\bar{o}_{(T+t)} \\
&\geq uT/(2c_A) + t - c_R T\bar{o}_{(T+t)} - c_R t\bar{o}_{(T+t)} \\
&= T(u/(2c_A) - c_R\bar{o}_{(T+t)}) + t(1 - c_R\bar{o}_{(T+t)}) \\
&\geq T(u/(2c_A) - 1/(4c_A)) + t(1 - 1/(4c_A)) \\
&\geq T/(8c_A) + t/2
\end{aligned}$$

where the second inequality follows from the fact that $\bar{o}_{(T+t)} \leq \bar{l} = 1/(4c_A c_R)$ since we are in an R phase. The last inequality follows from the facts that $u \geq 7/8$ and $c_A \geq 1$. (We choose "nice" constant rather than the tightest constants.)

Hence the competitive ratio at time $T+t$ is at most

$$\frac{T+t}{T/(8c_A) + t/2} \leq \max\{8c_A, 2\} = 8c_A = a_r$$

as needed for the first invariant.

6 Combining Admission Control Algorithms

In this section we briefly describe how to combine a collection of online algorithms into one master algorithm with a good acceptance competitive ratio and into another master algorithm with a good rejection competitive ratio. Results of this form already exist in the literature [2, 6, 7, 9] 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 which guarantees both rejection and acceptance competitiveness.

The main ingredient of the combining algorithms is switching between algorithms. Switching algorithms might means that we need to preempt some or all requests that we currently serve.

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 combining algorithms can be either randomized or deterministic.

6.1 Combining algorithms to minimize rejection

Given k (possibly preemptive) on-line algorithms, denoted by $R_1, R_2, \ldots, R_k$, we would like to construct an algorithm, which for any input sequence, competes with the best algorithm, among the k, for the given sequence. Specifically, for a sequence of requests σ let $R^*(\sigma) = \min_i R_i(\sigma)$. We construct a deterministic preemptive online combining algorithm REJ_{det} such that for any σ, we have $REJ_{det}(\sigma) = O(kR^*(\sigma))$. We also provide a randomized preemptive online algorithm REJ_{rand} which guarantees that

$$REJ_{rand}(\sigma) = O(R^*(\sigma) \log k) \ .$$

The deterministic algorithm REJ_{det} uses a simple greedy strategy. At time t, let $min(t) = \min\{R_i(\sigma_t)\}$. The algorithm REJ_{det} at time t uses one of the algorithms that achieve the minimum rejection, i.e. $min(t)$, and preempts all the requests the selected algorithm rejected or preempted. In the worse case REJ_{det} might reject $k \cdot min(t)$ requests up to time t, establishing the following theorem.

Theorem 3 *The deterministic algorithm REJ_{det} rejects at most $kR^*(\sigma)$ for any sequence σ of requests.*

The randomized algorithm REJ_{rand} uses simple doubling strategy. Initially, it accepts all requests as long as possible with no rejection and then set $\lambda = 1$. When the condition is violated it sets $\lambda \leftarrow 2\lambda$, choose a random i such that $R_i(\sigma) \leq \lambda$ and stick with it until the inequality is violated. (If such i does not exits then the condition is immediately violated and we double the value of λ.)

Since this problem can be viewed as a variant of the layered graph traversal [7, 9, 6] one can show the following:

Theorem 4 *The randomized algorithm REJ_{rand} rejects at most $O(\log k)$ times more requests than $R^*(\sigma)$ for any sequence σ of requests.*

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 $R_i(\sigma) \leq c_R opt(\sigma)$.

Corollary 5 *The deterministic algorithm REJ_{det} is $O(c_R k)$ competitive and the randomized algorithm REJ_{rand} is $O(c_R \log k)$ competitive.*

6.2 Combining algorithms to maximize acceptance

In this subsection we have the same scenario as in the previous section but the goal is to maximize the number of accepted requests. Given k algorithms $A_1, A_2, \ldots, A_k$ we would like to construct an algorithm which is as well as the best algorithm among the k for the given sequence. Specifically, for a sequence of requests σ let $A^*(\sigma) = \max_i A_i(\sigma)$. We will construct one randomized preemptive online algorithm ACC such that for any σ we have

$$ACC(\sigma) \geq A^*(\sigma)/\log k \ .$$

As before, we will 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 worse case we left with a single accepted call. This suggests that there is no deterministic competitive combining algorithm For this reason 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 1, while the others have a benefit of zero. 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 yield by the current option. Switching between option corresponds in our setting to switching between algorithms while possibly preempting all currently accepted requests. It is shown in [2] that using polylogarithmic number of switches, the decision maker, with high probability, has benefit which is at least $O(\log k)$ fraction of the benefit yield by the best choice. Therefore,

Theorem 6 *The randomized algorithm ACC accepts at least $O(\log k)$ fraction of requests compared with $A^*(\sigma)$ for any sequence σ of requests.*

As before 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 $A_i(\sigma) \geq opt(\sigma)/c_A$.

Corollary 7 *The algorithm ACC is $O(c_A \log k)$ competitive.*

7 Conclusions and open problems

We have described a procedure that given an algorithm A with competitive ratio c_A for benefit, and an algorithm R with competitive ratio c_R for cost, produces an online algorithm that simultaneously achieves competitive ratio $O(c_A^2)$ 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.

References

[1] R. Adler and Y. Azar. Beating the logarithmic lower bound: randomized preemptive disjoint paths and call control algorithms. In *Proc. 10th ACM-SIAM Symp. on Discrete Algorithms*, pages 1–10, 1999.

[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 *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 519–530, 1996.

[3] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive online routing. In *34th IEEE Symposium on Foundations of Computer Science*, pages 32–40, 1993.

[4] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. In *Proc. of 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 312–320, 1994.

[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 Found. of Comp. Science*, pages 412–423, 1994.

[6] Y. Azar, A. Broder, and M. Manasse. On-line choice of on-line algorithms. In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms*, pages 432–440, 1993.

[7] 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, Lecture Notes in Computer Science 318, Springer-Verlag, Berlin, 1988, 176–189. of Computer Science, October, 1987.

[8] A. Blum, A. Kalai, and J. Kleinberg. Admission control to minimize rejections. In *Proceedings of WADS 2001; LNCS 2125*, pages 155–164, 2001.

[9] A. Fiat, D. Foster, H. Karloff, Y. Rabani, Y. Ravid, and S. Vishwanathan. Competitive algorithms for layered graph traversal. In *Proc. 32nd IEEE Symposium on Foundations of Computer Science*, pages 288–297, 1991.