Chapter 1
Competitive Non-Preemptive Call Control

Baruch Awerbuch* Yair Bartal† Amos Fiat† Adi Rosén†

Abstract

We deal with randomized competitive algorithms for non-preemptive call control on tree-like switching networks. We give an optimal $O(\log n)$ competitive algorithm for non-preemptive call scheduling on trees. We then extend the problem to include variable call rates, call durations, and arbitrary call benefits, and obtain a polylog competitive algorithm. We also show that many similar algorithms for different problems that can deal with constant values of parameters such as rates and benefits can be transformed into randomized algorithms that can deal with varying values of the parameters.

Using randomization, this extends the work of Garay et al. on call control for the line network [GGKMY] to tree networks, while avoiding the preemption requirement, and while allowing arbitrary benefits and arbitrary rates. Alternately, this can be viewed as a generalization of the work of Awerbuch, Azar and Plotkin for throughput competitive routing [AAP], limited to trees, but without the limitation of requiring communication rates to be a small fraction of the link bandwidth.

In common to all our results is an algorithmic paradigm — "Classify and Randomly Select". This algorithmic tool seems useful for a variety of on-line problems in addition to those presented in this paper.

1 Introduction

The competitive study of on-line virtual circuit routing algorithms has been the subject of several recent papers [AAFPW, GG, GGKMY, AKPPW]. This paper follows this natural phenomenon and deals with the competitive analysis [ST] of on-line non-preemptive call control algorithms.

Competitive analysis has been extensively used for both "cost problems" and "benefit problems". Algorithms for cost problems deal with events and spend some commodity to deal with them (*e.g.*, the k-server problem). In benefit problems we view events as opportunities to gain some profit, subject to some set of constraints defined for the problem (*e.g.*, maximize the size of a matching). We say a cost problem has a competitive ratio of c if there exists an on-line algorithm A such that for all sequences of events σ the costs associated with A are no more than c times the costs of an optimal offline adversary to deal with σ, up to an additive term. Similarly, we say a benefit problem has a competitive ratio of c if there exists an on-line algorithm A such that for all sequences of events σ the benefits accrued by A while dealing with σ are no less than $1/c$ times the benefits gained by an optimal adversary while dealing with σ, up to an additive term. When considering a randomized on-line algorithm the cost incurred, or the benefit gained, by the algorithm is the *expected* such cost or benefit, where the expectation is taken over the random choices of the algorithm. We analyze our algorithms against an oblivious adversary [BBKTW].

We use a "Classify and Randomly Select" paradigm for all our algorithms, this paradigm has been used implicitly in previous competitive work (*e.g.*, [LT]). This tool for the construction of randomized competitive algorithms for benefit problems can best be illustrated by the following example.

Say you have one icebox that can contain one type of ice cream. You wander about the beach trying to sell ice cream and your goal is to sell as many cones as possible. If you know that the people on the beach only eat chocolate, vanilla, and strawberry, but you don't know what percentage eat what flavor then you would naturally toss a coin and choose amongst the three flavors with equal probability. You would be 3 competitive irrespective of what the actual percentages are. If, in addition, you know that some people eat sherbet, but that for every sherbet eater there is at least one customer that eats chocolate, vanilla, or strawberry, then you would be 6 competitive if you use the above algorithm.

Thus, the framework is as follows: You deal with on-line events, each of which offers some benefit. You might not be able to take advantage of all events so you

*Lab. for Computer Science, MIT. E-mail: `baruch@theory.lcs.mit.edu`.

†Department of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel. Supported in part by a grant from the Israel Science Foundation. E-mail: `yairb@math.tau.ac.il`, `fiat@math.tau.ac.il`, `adiro@math.tau.ac.il`.

predetermine a collection C of classes and assign each on-line event to one of these classes. Clearly, you must be able to assign each event to one of the classes in an on-line fashion.

You first select a class at random, and then consider only those events that get classified into that class. The selection could be done uniformly at random (as with the ice cream example above) or could be tailored more specifically if you have some additional knowledge on inter-classes benefit amortizations.

Our central result on non-preemptive virtual circuit routing is an optimal randomized on-line algorithm for call-control in tree networks. It uses a classification component that classifies the arriving calls into $O(\log n)$ classes, where n is the number of vertices in the network. This algorithm assumes uniform call benefits and rates, and infinite duration for the calls and achieves a competitive ratio of $O(\log n)$. Additional applications of the "Classify and Randomly Select" paradigm are used to deal with arbitrary call benefits, rates, and durations. The introduction of each additional varying parameter increases the competitive ratio by a factor of $O(\log M)$, where M is the ratio between the smallest and largest values of the parameter in the sequence of calls.

Here, one can ask, has this ratio to be known in advance? We first present algorithms for the case where the ratio is known (but the minimal and maximal values are not known). Using techniques of Lipton and Tomkins [LT] one can define appropriate probabilities and build a variant of the "Classify and Randomly Select" paradigm for the case where the ratio is not known. In this variant even the number of classes is not known in advance. The competitive ratio for this case is slightly larger than logarithmic in the ratio M.

1.1 Background Aspens at al. [AAFPW] deal with a cost problem, the goal is to minimize the bandwidth required to deal with an on-line set of connection requests on an arbitrary graph. This can be interpreted to mean the following: given that the optimal adversary can deal with all communication requests, the on-line algorithm can also deal with all requests but requires an $O(\log n)$ factor increase in the bandwidth capacities of the network links. In this work communication requests are of infinite duration. Azar et al. [AKPPW] get an $O(\log n)$ competitive algorithm within the same framework but allow communication requests to include a communication duration argument.

Awerbuch, Azar and Plotkin [AAP] deal with a benefit version of the problem, where the bandwidth limit of a link is considered as an absolute limit (or alternately, communication rate requirements cannot be forcibly reduced). This forces one to ignore some communication requests, and the on-line algorithm has to decide upon the arrival of each request whether to accept it or reject it. Every communication request includes a "benefit" argument and the goal is to maximize the benefit subject to the actual bandwidth limitations. In this framework they obtain an $O(\log n)$ competitive algorithm for their problem, but make one major simplifying assumption. They require that no single communications request can require more than $1/\log n$ of the bandwidth of a single link. The virtual circuit routing problem can be viewed as a form of "path packing" problem; obviously, a packing problem is much easier if only small items are allowed.

If rates can be large, *i.e.*, if a single call can completely saturate a link, and if the bandwidth limit of a link is an absolute limit, then the resulting benefit problems are known as *Call Control* problems [GGKMY, GG]. Calls are characterized by a pair of vertices, and optionally by a duration, a rate, and a benefit.

For call control problems one immediate question is "preemption or not?". Preemption means that an existing call, already communicating, can be discarded. If we don't allow preemption then we have to decide at the time the communication request arrives whether we are going to route the call for its full duration.

Clearly, allowing preemption may be annoying to the telephone subscriber but it is convenient for the phone company. More to the point, if the phone company does not gain anything from preempted calls then an optimal adversary will never use preemption. Thus, preemption can only help the on-line algorithm.

Garay et al. [GGKMY] give $O(\log n)$ competitive algorithms for preemptive call control when the network is a line. *I.e.*, there are nodes arranged along a line, calls connect two nodes and while a call is active then no other call can pass through the barrier defined by the active call. They consider a number of natural benefits for calls: unit price, distance, and duration, but they do not allow arbitrary call benefits.

Lipton and Tomkins [LT] do not deal with routing problems (connect v to u), but rather with scheduling tasks that require access to a nonsharable resource. This is described as scheduling intervals on a line. The competitive ratio is slightly more than logarithmic in the ratio between the maximal use duration and the minimal use duration. This ratio need not be known in advance.

1.2 Results We give non-preemptive call control algorithms. These are the first competitive algorithms for non-preemptive call control when the requested rate can reach the full bandwidth. We also give the first competitive algorithm that can deal with all problem parameters: rates, durations, and arbitrary benefits.

All routing algorithms above [AAFPW, AAP, AKPPW, GG, GGKMY] are *deterministic*. For non-preemptive call control it is not hard to see that deterministic algorithms cannot achieve a low competitive ratio. *E.g.*, for a line network with n vertices there is a lower bound of $\Omega(n)$ on the competitive ratio [AAP]. If we allow arbitrary benefits, we can show that a deterministic call control algorithm must have a nearly linear competitive ratio even if preemption is allowed.

Thus, our algorithms are all randomized. We start by giving an optimal $O(\log n)$ competitive algorithm for non-preemptive call control on trees. In this *base algorithm* we assume that calls are of infinite duration, that all call rates are one (which is the bandwidth of the links), and that the benefits of all calls are equal. Thus, the goal can be stated as maximizing the number of calls we can serve, while ensuring that every scheduled call is edge-disjoint from all other scheduled calls.

We build upon the base algorithm and devise an algorithm that handles the general case of arbitrary benefits, rates and durations. The introduction of each varying parameter increases the competitive ratio by a factor of $O(\log M)$, where M is the ratio between the smallest and largest value of the parameter. In fact, we introduce a paradigm to transform on-line algorithms (for benefit problems) that can handle events having almost uniform values of a certain parameter (such as benefit) into randomized algorithms that can handle varying values of this parameter. We also use this paradigm to show how to transform any algorithm for the base case (on any network) into an algorithm for the case of arbitrary benefits and durations, while increasing the competitive ratio by the above mentioned factor. Using techniques of Lipton and Tomkins [LT] this can be done also when the ratio is not known in advance; in this case the increase in the competitive ration is slightly greater than logarithmic.

One could consider a different model of a switching network when every switch can connect one input to one output, while disconnecting all other edges adjacent to the switch. In this context vertex-disjoint paths would be relevant. A simple modification of our base algorithm would give an optimal $O(\log n)$ competitive algorithm for the case of vertex-disjoint paths.

2 The Call-Control Problem

2.1 The Model A graph $G = (V, E)$ consisting of $|V| = n$ nodes represents a communication network where the nodes are communication switches and the edges are communication links. All communication links have *bandwidth* (capacity) of one unit, and thus can transfer at any specific time only calls whose accumulated *rate* is at most 1.

A call is represented by a tuple:

$$C = (v_s, v_t, r, t, d, b) ,$$

where v_s is the origin of the call, v_t is its destination, r is its rate, t is the starting time (or arrival time) of the call, d its duration, and b the benefit accrued by the algorithm if it accepts the call.

Calls are presented to a *call-control* algorithm in an on-line manner and the algorithm has, upon receipt of each call, to either *accept* or *reject* it. A call is accepted by assigning it to some path from v_s to v_t such that for each edge in the path the sum of rates of calls using that edge at time t is at most $1 - r$. Once a call is accepted it continues to use the path for d time units until completion (no preemption). For any accepted call the algorithm accrues the benefit b associated with the call. The objective is to maximize the total benefit gained.

In the rest of the paper we assume the graph G is a tree.

We denote the unique path p in the tree from vertex v to vertex u by (v, u). A path is also viewed as a set of edges. We may use the terminology "call" and "path" interchangeably, when possible, because all calls on a tree network define a unique path. This will be clear from the context.

We give the following definitions required later for the algorithm and proofs:

- Two paths p and q *intersect* iff $p \cap q \neq \phi$.
- Two paths p and q are *consistent* iff $p \cap q = \phi$.
- A set of paths is *consistent* if all paths in the set are pairwise consistent.

We first consider the base case where all call durations are infinite, all benefits associated with the calls are equal (without loss of generality assume they are 1), and all rates are 1. In this case a call is defined by a pair $C = (v_s, v_t)$, since all other values are implicit. Note that when such a call is accepted it totally blocks the edges along the path that it defines. We later build upon this base algorithm and extend it to deal with the case where all parameters may vary.

3 The Base Algorithm

In this section we give the randomized algorithm for the base case. The randomized algorithm classifies the calls on the tree into several classes. It then randomly selects one of the classes, and later considers only those calls that fall into the selected class, rejecting all other calls.

In the preliminary version of this paper we presented a bookkeeping algorithm that classifies in an on-line manner the arriving calls into $O(\log n)$ classes of consistent calls. Subsequent to our work, Blum, Fiat, Karloff and Rabani [BFKR] devised an algorithm for the related problem of deterministic preemptive call-control on trees. Here we modify their deterministic preemptive algorithm and use this modification in place of our original classification algorithm in order to classify the calls into $O(\log n)$ classes such that within each class the greedy algorithm achieves a constant competitive ratio. Since the new classification algorithm is considerably simpler and achieves the same competitive ratio as our original algorithm, we present it in what follows.

3.1 The Classification Algorithm To define the classification we assign a number to each of the nodes in the following recursive way. Given a tree T of k nodes there always exists a vertex v such that its removal leaves a forest of trees, each of which contains at most $2/3k$ nodes. Such a node always exists as trees have 2/3-separators of size 1 (cf. [L]). If fact, it is not hard to see that there always exists a vertex such that its removal splits a tree into subtrees of size at most $k/2$.

We associated levels with the vertices recursively, starting with level $l = 1$. We choose a vertex v as described above, assign level l to v and apply the procedure recursively to each of the subtrees, starting at level $l+1$. Thus each of the nodes of the tree is assigned a level in the range $1 \leq l \leq \log n$. We now assign a level to each of the edges, by giving it the lower level among the levels assigned to its two endpoints.

The classify algorithm can now be stated in a very simple way:

Let l be the lowest-numbered level among the levels of the edges that are included in a call. Then, classify the call into class number l.

The above rule classifies the calls into $O(\log n)$ disjoint classes.

We now argue that if we consider only the calls of a single of the above defined classes, then the greedy on-line algorithm is 2-competitive.

Claim 3.1 *Let p be call of class l. Then there are at most 2 edges of level l included in p.*

Proof. If a path on the tree includes more than 2 edges of level l, then this path connects two nodes of level l. However, every such path must pass through a node of a lower-numbered level (due to the nested separation nature of the levels). Thus the path in question would not have been of level l. □

Claim 3.2 *Let $\mathcal{P}$ be a set of paths of the same level l sharing an edge. Then, all the paths of $\mathcal{P}$ share a level l edge.*

Proof. Each path $p \in \mathcal{P}$ includes a level l edge, which is adjacent to a level l node. Denote this node by v_p.

If all these nodes are not the same node in the tree, then there are two paths p and q in $\mathcal{P}$ such that p and q establish a route in the tree that connects v_p and v_q, both level l nodes. By the same arguments as those in the proof of the previous claim, it follows that either p or q are not of level l.

Thus, all paths in $\mathcal{P}$ are adjacent to the same node of level l, and as they all share an edge, they must also share an edge adjacent to this node. All edges adjacent to a level l node have level at most l, and since all paths in $\mathcal{P}$ are of level l, they can include of those only the edges of level l. Thus, the paths in $\mathcal{P}$ share an edge of level l. □

Lemma 3.1 *The greedy on-line algorithm applied to the calls of a single class is 2-competitive.*

Proof. Let p be a call not accepted by the greedy algorithm, that is, when it arrives there is already a call q that was accepted by the greedy algorithm, and intersects p. Then, by Claim 3.2 p and q share an edge of level l. We charge the call p to this level l edge of q. Since each such edge can accommodate only one consistent call, and as by Claim 3.1 there are at most two such edges per accepted call, we conclude that there are at most 2 consistent non-accepted calls per accepted call. The lemma follows. □

3.2 The Randomized Base Algorithm Let $N \leq \log n$ be the maximum level number assigned to the the edges of the tree. The randomized call-control algorithm selects with probability $\frac{1}{N}$ one of the classes defined by the levels, and applies the greedy algorithm to the calls that falls into this class, ignoring all other calls.

Theorem 3.1 *The Base Algorithm is $O(\log n)$-competitive.*

Proof. Let $\mathcal{OPT}$ denote the optimum set of calls that an (off-line) algorithm can accept. Let C_i be the set of calls that fall into the class of level i and $OPT(C_i)$ be the maximum size consistent subset of C_i. Then,

$$|\mathcal{OPT}| \leq \sum_{i=1}^{N} |OPT(C_i)| \ .$$

The expected number of calls accepted by the base algorithm is,

$$E(ALG) \geq \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} |OPT(C_i)| \ .$$

As $N \leq \log n$, the base algorithm is $O(\log n)$-competitive. □

4 Lower Bounds

Before starting our discussion of lower bounds, we distinguish between *strictly competitive* ([BFR]) lower bounds and competitive lower bounds. An algorithm is called strictly competitive with a competitive ratio of c if the algorithm's benefit (or cost) is within a factor of c of optimal. In the definition of a competitive algorithm, we usually allow an additive term.

For finite games such as non-preemptive call control for calls of infinite duration (or the on-line Steiner tree problem [BFR]), it does not really make sense to allow an additive term. If such an algorithm is to be used as a building block for more complex situations (such as calls with duration), this additive term would multiply and the final result would not be competitive.

While our randomized algorithms are strictly competitive, and our lower bounds are given for strictly competitive randomized algorithms, we can prove most of the randomized lower bounds even if we omit the strict competitiveness restriction.

We give several lower bounds in this section, dealing with the base problem described in the previous section and with several extensions of the problem including arbitrary durations and benefits. The next section includes algorithms that deal with these extensions of the base problem.

We first show that our base algorithm for trees is optimal up to a constant factor even when the problem is restricted to a line network.

We next consider the problem where calls may have different durations. Let T be the ratio between the largest and smallest possible durations. We show a lower bound of $\Omega(\log T)$ for any randomized on-line call control algorithm even when the network is a single link.

Both these proofs are similar and make use of a simple combinatorial lemma (Lemma 4.1).

We then deal with the case where calls may have different benefits and observe that any randomized competitive ratio must be $\Omega(\log M)$ where M is the ratio between the highest and lowest possible benefits. This holds even if the network is a single link.

Next, we show that dealing with arbitrary benefits deterministically requires a lower bound of $\Omega(M)$ on the line network even if preemption is allowed, if M is $O(n/\log n)$ where n is the number of nodes on the line. Therefore, the results of [GGKMY] cannot be generalized to deal with arbitrary benefits. This, combined with our upper bound in the next section, shows an exponential improvement in the competitive ratio for randomized algorithms.

We now turn to prove the lower bounds.

Given a randomized on-line algorithm A for a benefit problem with a competitive ratio c, we show how to bound c from below using a construction called an *adversary tree*. This is applicable for problems such that every event can be either accepted or rejected, and a benefit of one is attained for every accepted event.

An adversary tree of depth k is a full binary tree of depth k that associates events with the vertices of the tree. The root is at depth 0 and the leaves at depth k. The problem constraints and the event structure have the property that only one event can be accepted among the set of events along any path from the root to a leaf. Any set of events such that no two events are along the same path from the root to a leaf can be accepted in whole.

The probability distribution of the on-line algorithm configuration space determines probabilities associated with vertices of this tree. The probability associated with a vertex is determined when the associated event occurs. These probabilities are absolute probabilities, *i.e.*, the probability associated with vertex v is the probability that v is accepted, while all its ancestors and descendants are rejected.

Given an on-line algorithm A, the adversary prunes the tree T while considering the probability distribution of the on-line algorithm configuration. The tree T is traversed recursively in preorder. The adversary decides to recurse on the children of a vertex v only if the sum of probabilities from the root to v is at least $\ell/2c$ where ℓ is the number of vertices on the path from the root to v. Otherwise the adversary concludes that v is a leaf in the pruned tree.

It now follows that the final tree has the following two properties:

1. The average probability associated with vertices along any path from the root to any internal vertex is at least $1/2c$.

2. The average probability associated with vertices along any path from the root to a leaf of depth less than k is less than $1/2c$.

Lemma 4.1 *Given an adversary tree of depth k for a certain problem, it follows that the strictly competitive ratio c of any on-line randomized algorithm for the problem satisfies $c = \Omega(k)$.*

Proof. If there is some leaf v of the original adversary tree that remains in the pruned tree and the average probability along the path from the root to v is at least $1/2c$, it follows that the sum of probabilities along this path is at least $(k+1)/2c$. As events along this path are mutually exclusive, this sum is no more than 1 and thus $c \geq (k+1)/2$.

We now show that such a leaf must exist.

If no such leaf exists, then we may assume that

1. The average probability associated with vertices

along any path from the root to any internal vertex is at least $1/2c$.

2. The average probability associated with vertices along any path from the root to a leaf is less than $1/2c$.

We seek to show that the average probability of all vertices in the tree is less than $1/2c$. We partition the tree into disjoint paths of vertices, each of which has an average probability less than $1/2c$, this proves the claim.

To see that this can be done, we first mark all vertices along a path from the root to a leaf in the pruned tree. The average probability of the vertices marked is less than $1/2c$. Consider another path from the root to an unmarked leaf in the pruned tree. The average probability of the vertices along this path is also less than $1/2c$, but some prefix of this path may have been already marked; this prefix has an average probability of at least $1/2c$. Thus, the unmarked vertices along this path, forming a suffix of the path from the root to a leaf, have an average probability less than $1/2c$. We now mark these vertices. As all vertices will be eventually marked, we conclude that the average probability of all vertices in the tree is less than $1/2c$.

If the adversary presents the sequence obtained from the preorder traversal of the pruned tree, then its benefit is maximized by accepting all events associated with the leaves of the tree. The total number of events in the tree is strictly less than twice the number of leaves. The expected on-line benefit is less than the number of events times $1/2c$. Thus, we conclude that the on-line benefit is strictly less than $1/c$ times the adversary benefit, a contradiction. □

Theorem 4.1 *There is a lower bound of* $\Omega(\log n)$ *on the competitive ratio of any randomized call-control algorithm for calls of* equal benefit *on the* line *with* n *nodes.*

Proof.

We describe an adversary tree of depth $\Omega(\log n)$.

We assume $n = 2^k + 1$ for some $k > 0$.

Let the nodes on the line be named $1, 2, \ldots, 2^k + 1$, and denote by (v, u) a call from node v to node u.

The root of the adversary tree is the call $(1, 2^k + 1)$.

The two children of the root are the two calls: $(1, 2^{k-1} + 1), (2^{k-1} + 1, 2^k + 1)$.

In general, the nodes at depth i, $0 \le i \le k$, are associated with the calls $((j-1) \cdot 2^{k-i} + 1, j \cdot 2^{k-i} + 1)$, for $1 \le j \le 2^i$, where every such call is a child of a parent call that is inconsistent with it.

This adversary tree fits the conditions assumed in Lemma 4.1, and thus guarantees a lower bound of $\Omega(\log n)$. □

Very similarly, we give a lower bound logarithmic in the ratio T between the shortest and longest possible durations on a network that consists of a single edge.

Theorem 4.2 *There is a lower bound of* $\Omega(\log T)$ *on the competitive ratio of any randomized call-control algorithm for calls of* arbitrary durations *on a* single edge.

Proof Sketch. Let $k = \log T$. The root of the tree is an event to be released immediately with duration 2^k. The left child of the root is to be released immediately with duration 2^{k-1}, the right child of the root is to be released at time 2^{k-1} and of duration 2^{k-1}. The rest of the tree is constructed recursively until a depth of k. We note that the preorder traversal respects the release time order of the events. □

We now consider the problem when calls may have different benefits. Let M be the ratio between the largest and smallest possible benefits.

Theorem 4.3 *There is a lower bound of* $\Omega(\log M)$ *on the competitive ratio of any randomized call-control algorithm for calls of* arbitrary benefit *on a* single edge.

The proof follows by generating a sequence of calls of exponentially increasing benefits.

The next theorem states that for deterministic algorithms there is a much higher lower bound even if preemption is allowed.

Theorem 4.4 *There is a lower bound of* $\Omega(M)$ *on the competitive ratio of any deterministic call-control algorithm for calls of* arbitrary benefit *on the* line *of* n *nodes, if* $M \le n/2 \ln n$*, even if preemption is allowed.*

Proof. Let the nodes on the line be: $1, 2, \ldots, n$. The adversary first gives a call $(1, n)$ of benefit M. This call must be accepted by any algorithm. Next the adversary gives a sequence of calls $c_1, c_2, \ldots, c_{n-1}$, where $c_l = (l, l+1)$ has benefit 1 if $l < M$ and benefit $b_i = (1 + \frac{1}{M-1})^i$ for $i = l - M, (0 \le i < n - M)$. If the on-line algorithm performs a preemption the adversary stops the sequence.

Now, if the on-line algorithm preempts when one of $c_1, c_2, \ldots c_{M-1}$ arrive then its profit equals 1, while the adversary's profit is M. Otherwise if the preemption is done upon receiving call $c_{M+i}, (0 \le i < n - M)$ then the on-line profit is b_i while the adversary would accept all calls $c_1, c_2, \ldots, c_i$ accruing a profit:

$$(M-1) + \sum_{0 \le j \le i} b_j =$$

$$(M-1) + \sum_{0 \le j \le i} (1 + \frac{1}{M-1})^j =$$

$$(M-1) + (M-1) \cdot ((1 + \frac{1}{M-1})^{i+1} - 1) =$$

$$(M-1)\cdot b_i \cdot (1+\frac{1}{M-1}) = M \cdot b_i$$

If the on-line algorithm never preempts then its benefit is M while the offline adversary can accrue a benefit of $M \cdot b_{n-1-M}$. Thus, we need that $b_{n-1-M} \geq M$ in order to guarantee the lower bound of M in this case too. This leads to the limitation on the value of M. □

5 The General Case

5.1 Varying Parameters in Benefit Problems

We consider on-line benefit problems of the following form: Entities are presented to the on-line algorithm and it has to either accept them or reject them, under some consistency (independence) constraints defined by the problem. The algorithm accrues some benefit associated with the entity if it is accepted, and does not accrue any if it is rejected. The benefit need not be equal for all entities. The aim of the algorithm is to maximize the benefit gained.

We describe a "Classify and Randomly Select" method to eliminate the effect, on the performance of the algorithm, of some parameter p of the entities, given that the algorithm can tolerate different values of this parameter up to a factor of 2. That is, given a c-competitive algorithm that is suitable for the case that parameter p assumes values that differ by at most a factor of 2, we build upon it and devise an $O(c \log M)$-competitive randomized algorithm that tolerates values that differ by a factor of at most M, i.e., M is the ratio between the smallest and the largest value p can get.

Let $\mathcal{A}$ be an algorithm tolerating values of p that differ by a factor of at most 2. We define a randomized algorithm $\mathcal{A}'$. The algorithm will create in an on-line manner at most $L = \lceil \log M \rceil + 1$ classes $\mathcal{G}_i$, $1 \leq i \leq L$, and will associate with each class a number g_i. At each time the algorithm holds a value $1 \leq k \leq L$, and a set of classes $\mathcal{G}_i$, $1 \leq i \leq k$; at the beginning $k = 0$. Upon the arrival of a new entity E with parameter p such that $2^{j-1} < p \leq 2^j$, the algorithm checks if there is a class $\mathcal{G}_i$ such that $g_i = j$. If there is such a class then the algorithm associates entity E with this class. Otherwise $\mathcal{A}'$ increments k by 1, and creates a new class $\mathcal{G}_k$ with $g_k = j$ and associates E with it. When a new class is created then if $\mathcal{A}'$ did not yet select any of the classes $\mathcal{G}_i$ for $0 \leq i \leq k$ then it chooses to select the new class so that the absolute probability it is selected is $\frac{1}{L}$. Thus, the randomized algorithm $\mathcal{A}'$ selects one class among the classes $\mathcal{G}_i$. It uses algorithm $\mathcal{A}$ presenting to it those entities that fall into the selected class, and while ignoring all other entities.

Theorem 5.1 *If $\mathcal{A}$ is a c-competitive algorithm that tolerates values of p that differ by at most a factor of 2, then $\mathcal{A}'$ is an $O(c \log M)$-competitive randomized algorithm that tolerates values of p that differ by at most a factor of M.*

Proof. Let $OPT(G)$ denote the set of entities accepted by the optimal (off-line) algorithm when presented with a sequence G of entities. Denote by $B()$ the benefit associated with an entity or a set of entities (which is the sum of the benefits associated with the entities in the set).

Let G be the sequence of entities presented to $\mathcal{A}'$ and G_i the subsequence of G consisting of those entities which are in $\mathcal{G}_i$. Then $OPT(G) \subseteq \cup_i OPT(G_i)$, and $B(OPT(G)) \leq \sum_{i=1}^{L} B(OPT(G_i))$. Let $A(G_i)$ be the set of entities accepted by $\mathcal{A}$ when presented with the subsequence of entities G_i.

The expected benefit accrued by $\mathcal{A}'$ is

$$E(\mathcal{A}') = E(B(\mathcal{A}'(G))) = \sum_{i=1}^{L} \frac{1}{L} B(A(G_i)) \ .$$

Since $\mathcal{A}$ is c-competitive on the sequence of entities G_i, for any i, we have

$$E(\mathcal{A}') \geq \sum_{i=1}^{L} \frac{1}{L}\frac{1}{c} B(OPT(G_i)) \geq \frac{1}{cL} B(OPT(G)) \ .$$

□

Using techniques of Lipton and Tomkins [LT] the above paradigm can be modified to suit the case where the ratio M is not known in advance. Define $f(\epsilon) = \sum_{i=1}^{\infty} \frac{1}{i^{1+\epsilon}}$. $f(\epsilon) < \infty$ for all $\epsilon > 0$. When a new class $\mathcal{G}_k$ is first created and if $\mathcal{A}'$ did not yet select a class, then the algorithm now chooses to select $\mathcal{G}_k$ with probability such that the absolute probability it is selected is $p_k = \frac{1}{f(\epsilon)}\frac{1}{k^{1+\epsilon}}$ (instead of $\frac{1}{L}$). Since $\sum_{i=1}^{\infty} p_i = 1$ these are valid probabilities. Note that although the number L is not known to the algorithm, at most L classes can be created.

Theorem 5.2 *If $\mathcal{A}$ is a c-competitive algorithm that tolerates values of p that differ by at most a factor of 2, then $\mathcal{A}'$ is a randomized algorithm that achieves a competitive ratio of $O(c \log^{1+\epsilon} M)$, where M is the ratio between the largest and smallest values that p assumes. The value M need not be known to $\mathcal{A}'$.*

Proof. We follow and modify the proof of the previous theorem.

The expected benefit accrued by $\mathcal{A}'$ in the present case is

$$E(\mathcal{A}') = E(B(\mathcal{A}'(G))) = \sum_{i=1}^{L} p_i B(A(G_i)) \geq$$

$$\sum_{i=1}^{L} p_L B(A(G_i)) \ .$$

Since $\mathcal{A}$ is c-competitive on the sequence of entities G_i, for any i, we have

$$E(\mathcal{A}') \geq \sum_{i=1}^{L} p_L \frac{1}{c} B(OPT(G_i)) \geq$$
$$\frac{1}{c}\frac{1}{f(\epsilon)}\frac{1}{L^{1+\epsilon}} B(OPT(G)) \ .$$

□

5.2 Different Benefits and Durations We use this method to extend the base algorithm to the case where the calls have varying benefits associated with them, at the cost of augmenting the competitive ratio by a factor of $O(\log M)$, where M is the ratio between the smallest and the largest benefit.

Corollary 5.1 *Any c-competitive call-control algorithm for the case of equal benefits can be transformed into an $O(c\log M)$-competitive randomized algorithm for the call-control problem with calls with arbitrary benefits. If the ratio M is not known in advance then the competitive ratio is $O(c\log^{1+\epsilon} M)$.*

We can also handle the case where the calls have a starting time and a duration, which is known to the on-line algorithm at the time the call arrives.

Let T be the ratio between the longest and the shortest durations. Using Theorem 5.1 we have that we can use a competitive call-control algorithm that handles durations that vary by a factor of at most 2, to get a randomized algorithm that handles arbitrary durations with an $O(\log T)$ increase in the competitive ratio.

We now show that any call-control algorithm that assumes infinite durations can be transformed into an algorithm to handle durations that vary by a factor of at most 2, augmenting the competitive factor by only a constant factor. Let all durations d be such that $t^* < d \leq 2t^*$. Divide the time axis into slots of length t^*, and number the slots with increasing numbers. Choose with equal probability a number $q \in \{0,1,2\}$, and mark all slots numbered n such that n is congruent to q modulo 3 as *relevant* slots. The algorithm considers only calls with starting times in a relevant slot, ignoring all other calls. If we consider only calls that arrive in such one slot, then no call terminates before another arrives; therefore, the case is the same as the case of infinite durations. Furthermore, every call that starts in a relevant time slot terminates before the beginning of the next relevant time slot, thus every such time slot is handled separately.

Corollary 5.2 *Any c-competitive call-control algorithm that handles infinite durations can be transformed into an $O(c\log T)$-competitive randomized algorithm for the call-control problem with calls of arbitrary durations. If the ratio T is not known in advance then the competitive ratio is $O(c\log^{1+\epsilon} T)$.*

5.3 Different Rates To devise an algorithm that handles the case of calls with varying rates we employ again the above defined method. Let F be the ratio between the largest and the smallest rates. Using Theorem 5.1 we can use a call-control algorithm that can tolerate rates that vary by a factor of at most 2.

In what follows we show that our algorithm for the base case is $O(\log n)$ competitive even if the rates vary by a factor of at most 2. To do that we prove a stronger version of Lemma 3.1 that is applicable in that case.

Lemma 5.1 *If all calls have rate r such that $\alpha < r \leq 2\alpha$, for some α, then the greedy on-line algorithm, applied to the calls of a single class l defined in the base algorithm, is 6-competitive.*

Proof. Let p be a call of rate r not accepted by the greedy algorithm. That is, there is an edge $e \in p$ such that when p arrives the sum of the rates of the accepted calls that include e is greater than $1-r$. Consider the accepted calls that include e, and p. They are all of the same level l and they share an edge. By Claim 3.2 they also share a level l edge. Charge p to this edge. There can be at most $\lceil 1/\alpha \rceil - 1$ non accepted consistent calls that include this edge. On the other hand if a call is not accepted and charged to this edge, then there are already at least $\lceil 1/2\alpha \rceil - 1$ accepted calls that include this level l edge. As each call includes at most 2 level l edges, we conclude that the greedy algorithm is 6-competitive. □

Corollary 5.3 *There is an $O(\log n \log F)$-competitive randomized algorithm for the call control problem on trees for calls with arbitrary rates. If the ratio F is not known in advance then the competitive ratio is $O(\log n \log^{1+\epsilon} F)$.*

We note that all three extensions can be combined into one algorithm that handles the general case.

6 Acknowledgments

We are very grateful to A. Blum, H. Karloff and Y. Rabani for their kind permission to use the separator based algorithm for a shorter and simpler classify and randomly select algorithm. We are also very grateful to R. Lipton and A. Tomkins for their permission to cite their preliminary manuscript. We thank P. Berman for the many helpful suggestions he made concerning the presentation of the results.

References

[AAFPW] J. Aspenes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with application to machine scheduling and virtual circuit routing. In *Proc. 23rd Annual ACM Symposium on Theory of Computing*, pages 623–631, May 1993.

[AAP] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive on-line routing. In *Proc. 34th Symposium on Foundations of Computer Science*, November 1993.

[AKPPW] Y. Azar., B. Kalyanasundaram, S. Plotkin, K. Pruhs and O. Waarts. Online load balancing of temporary tasks. In *Proc. 3rd Workshop on Algorithms and Data Structures*. pages 119–130, August 1993.

[BBKTW] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. In *Proc.of the 22nd Ann. ACM Symp. on Theory of Computing*, pages 379–386, May 1990.

[BFKR] A. Blum, A. Fiat, H. Karloff and Y. Rabani. Private communication.

[BFR] Y. Bartal, A. Fiat, and Y. Rabani. Competitive Algorithms for Distributed Data Management. In *Proc. of the 24th Ann. ACM Symp. on Theory of Computing*, pages 39–48, May 1992.

[FKLMSY] A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, and N. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[GG] J.A. Garay and I.S. Gopal. Call preemption in communications networks. In *Proc. INFOCOM '92*. 1992.

[GGKMY] J. Garay, I.S. Gopel, S.Kutten, Y.Mansour, and M.Yung. Efficient on-line call control algorithms. In *Proc. 2nd Israel Symposium on Theory of Computing and Systems*, pages 285–293, June 1993.

[L] J. Van Leeuwen, Editor. Handbook of Theoretical Computer Science, Vol. A, Algorithms and Complexity. The MIT Press, 1990.

[LT] R.J. Lipton, and A. Tomkins. Online Interval Scheduling. *These Proceedings*.

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