Chapter 1
On-line Choice of On-line Algorithms

Yossi Azar * Andrei Z. Broder* Mark S. Manasse*

Abstract

Let $\{A_1, A_2, \ldots, A_m\}$ be a set of on-line algorithms for a problem P with input set I. We assume that P can be represented as a metrical task system. Each A_i has a competitive ratio a_i with respect to the optimum off-line algorithm, but only for a *subset* of the possible inputs such that the union of these subsets covers I. Given this setup, we construct a generic deterministic on-line algorithm and a generic randomized on-line algorithm for P that are competitive over *all* possible inputs. We show that their competitive ratios are optimal up to constant factors. Our analysis proceeds via an amusing card game.

1 Introduction

A common trick of the trade in algorithm design is to combine several algorithms using round robin execution. The basic idea is that, given a set of m algorithms for a problem P, one can simulate them one at a time in round robin fashion until the fastest of them solves P on the given input. It is easily seen that round robin execution is optimal among deterministic combining algorithms that have no specific knowledge of the problem domain and input. (As an aside, we show in Appendix A that randomization helps very little in this context; for any randomized combining scheme, and any $\epsilon > 0$, there is an input set of algorithms, such that the expected cost is greater than $(m - \epsilon)$ times the minimum cost, versus m times the minimum cost for round robin execution.)

For on-line algorithms the situation is more complicated: We are given a set $S = \{A_1, A_2, \ldots, A_m\}$ of on-line algorithms (deterministic or randomized – see below) for a problem P with input set I. Each algorithm A_i has a known competitive ratio a_i with respect to the optimum off-line algorithm, but only for a *subset* of the possible inputs, such that the union of these subsets covers I. We assume that P can be represented as a metrical task system (see [2] for definitions). Our goal is to construct an on-line algorithm for P that is competitive over *all* possible inputs.

Again, we are interested in algorithms that have no specific knowledge of the problem domain and input. More precisely, let σ_t be the sequence of requests up to time t. Let $C_i(\sigma_t)$ and $c_i(\sigma_t)$ be the configuration (respectively the cost) associated to A_i serving σ_t. At time t, the configuration associated to the combining algorithm must be one of the $C_i(\sigma_t)$'s. The decision to switch from $C_i(\sigma_{t-1})$ to $C_j(\sigma_t)$ can be based only on the values $c_i(\sigma_{t'})$ for $1 \le i \le m$ and $0 \le t' \le t$ and on no other domain or input specific information.

If the algorithms A_i and the combining construction are deterministic, then we call the new algorithm *on-line combine* and denote it $\textsc{min}_S$. If the construction uses random bits, we call the algorithm *randomized on-line combine*, denoted $\textsc{rmin}_S$. In this later case the A_i's might be randomized as well.

Fiat *et al.* [3] addressed the question of on-line combine in a context restricted to paging algorithms. Fiat, Rabani, and Ravid [5] considered the general case and showed that constructing a $\textsc{min}_S$ algorithm for an arbitrary set of on-line algorithms is equivalent to the *layered graph traversal* problem analyzed by Papadimitriou and Yanakakis [6] and Baeza-Yates, Culberson, and Rawlins [1]. Using the results of these analyses, they obtained a $\textsc{min}_S$ algorithm with competitive ratio $O(m \cdot \max_i\{a_i\})$, which is optimal up to a constant factor when all the a_i's are equal, but not in general. In this paper we completely solve the general case, when the a_i's are arbitrary. This immediately yields a better competitive ratio for the k-server algorithm of [5].

In this paper, we show that the problem of combining on-line algorithms is equivalent, up to a small constant factor, to finding the value of a very simple two-player card game[1]. In this game, two identical decks of cards are given to two players. Simplifying slightly, the first player (corresponding to the on-line combining algorithm) places a card face-down on the table. The second player (the adversary) chooses a card from his hand, and turns it face up on the table. The first player

*DEC Systems Research Center, 130 Lytton Ave. Palo-Alto, CA 94301. E-mail: azar@src.dec.com, broder@src.dec.com, msm@src.dec.com .

[1]Field experiments show that 5-year olds can easily play it...

then exposes the matching card, either from her hand, in which case no score is recorded, or by showing the card on the table, in which case the second player wins the value of the card. The pair is removed from play, and the players play another round with the reduced decks, until both players run out of cards.

A few variations exist, all of which turn out to be equivalent in terms of optimal strategy: the first player can be required to pick the same card to place face down for each round until it is matched by the second player, or not; the second player is required to arrange the order in which he will play cards in every round before play commences; one can even require the first player to select a schedule in advance for which card she will place face down next, except for those cards that are matched before their turn comes. It turns out that optimal play for all these variants results in the same total or expected score for the second player. In the deterministic case, the total is clearly the value of all the cards, since the second player can inspect the first player's strategy, and play cards in exactly the same order.

The randomized case has the following optimal strategy for building a schedule: let each card have a probability proportional to the inverse of its value, and choose a card using that distribution. That card is the last card in the schedule. Repeat this procedure on the remaining cards to find the schedule in reverse order.

By analyzing the card game, we obtain the following results:

THEOREM 1.1. *Let $S = \{A_1, A_2, \ldots, A_m\}$ be a set of deterministic on-line algorithms for a metrical task system P with input set I. Assume that each A_i has a competitive ratio a_i with respect to the optimum off-line algorithm for a subset of the possible inputs such that the union of these subsets covers I. Then there exists a deterministic* MIN$_S$ *algorithm with competitive ratio $O(\sum_{1\le i\le m} a_i)$, and no deterministic on-line algorithm can do better in general, except for a constant factor.*

The improvement with respect to the previous bounds is relevant when the average of the a_i's is substantially smaller than their maximum. In particular, our MIN$_S$ algorithm reduces the competitive ratio of the k-server algorithm of [5] by a factor of $k!/2^{O(k)}$. (See section 6.)

For the randomized case we need first to discuss a function that will play an important role in what follows. Let $a_1, a_2, \ldots$ be a sequence of positive numbers. For any set T of natural numbers we define $f(T)$ by the recurrence

$$f(\emptyset) = 0$$
$$f(T) = \frac{1 + \sum_{i\in T} f(T \setminus \{i\})/a_i}{\sum_{i\in T} 1/a_i}, \qquad T \neq \emptyset. \tag{1.1}$$

Let $[m]$ stand for the set $\{1, 2, \ldots, m\}$. Note that $f([m])$ is a symmetric rational function of $a_1, \ldots, a_m$. In particular

$$f(\{1\}) = a_1,$$
$$f(\{1,2\}) = \frac{a_1^2 + a_2^2 + a_1 a_2}{a_1 + a_2},$$

but the numbers of terms grows very fast: $f(\{1,2,3\})$ has 19 terms, and $f(\{1,2,3,4\})$ has 390. Nevertheless, we can crudely bound $f(T)$ by

$$H_m \min_{i\in T} a_i \le f(T) \le H_m \max_{i\in T} a_i, \tag{1.2}$$

where $m = |T|$, and H_m is the m'th harmonic number. Better but more complex bounds will be presented in Section 3.1.

Now we can state our result for randomized on-line combine.

THEOREM 1.2. *Let $S = \{A_1, A_2, \ldots, A_m\}$ be a set of deterministic or randomized on-line algorithms for a metrical task system P with input set I. Assume that each A_i has a competitive ratio a_i with respect to the optimum off-line algorithm for a subset of the possible inputs such that the union of these subsets covers I. Then there exists a randomized* RMIN$_S$ *algorithm with competitive ratio $O(f([m]))$, and no randomized on-line algorithm can do better in general, except for a constant factor.*

Plugging equation (1.2) into the theorem yields the weak upper bound $O(\log n \cdot \max_i a_i)$ which was obtained in [4].

2 The layered graph traversal problem

This problem was introduced and analyzed in [1] and [6].

A *layered graph* is an undirected graph with the property that its vertices can be divided into layers $L_0, L_1, L_2, \ldots$, such that all edges run between consecutive layers. Each edge e, has a certain non-negative length $l(e)$. A *disjoint-paths* layered graph consists exactly of m paths with a common first vertex s, called the *source*, but otherwise vertex disjoint. Thus, the graph can be divided into layers $L_0 = \{s\}, L_1, L_2, \ldots$, such that layer i for $i > 0$ consists of the m vertices that are i edges away from the source on each path.

An on-line layered graph traversal (LGT) algorithm starts at the source and moves along the edges of the graph. Each time it moves along an edge (in any direction), it pays a cost which is the length of the edge. Its goal is to reach a *target* which is a vertex in the last layer. The lengths of the edges between layer L_{i-1} to L_i are revealed to the algorithm only when a vertex in L_{i-1} is reached for the first time. (The lengths do not change over time.) The target vertex becomes known only when the algorithm reaches a vertex in the next-to-last layer.

The competitive ratio of the on-line traversal algorithm is the worst case ratio between the distance traveled by the on-line algorithm and the length of the shortest path from the source to the target. (For disjoint paths graphs, this path is unique. Also in this case any LGT algorithm must advance one layer at a time either by continuing on its current path, or by backtracking to the source and choosing a different path.)

For general layered graphs, the competitive ratio is exponential for deterministic algorithms, but polynomial for randomized ones [4, 7]. For disjoint-paths layered graphs the optimal deterministic algorithm has competitive ratio $1 + 2m(1 + \frac{1}{m-1})^{m-1} \approx 2em$ (see [6] and [1]). For randomized algorithms Fiat *et al.* [4] showed that the competitive ratio is $\Theta(\log m)$.

For the remainder of this paper we will consider only disjoint-paths layered graphs. We need a slight generalization of the model above: we assume that each path P_i has a known associated *waste factor* a_i. For each edge e on P_i, the off-line algorithm pays $l(e)/a_i$, while the on-line algorithm pays $l(e)$ as before. Thus the competitive ratio becomes a function of $a_1, \ldots, a_m$, and the preceding model corresponds to $a_1 = a_2 = \cdots = a_m = 1$.

Following [5] we show now that constructing a MIN_S algorithm is equivalent to an algorithm for the modified LGT with the same competitive ratio.

First assume that a (modified) LGT algorithm is given. To construct a MIN_S algorithm, we construct a disjoint-paths layered graph which associates a path P_i with each algorithm A_i. We set the waste factors to be the competitive ratios $a_1, \ldots, a_m$ and simulate $A_1, \ldots, A_m$ on the sequence of requests as follows.

When a request is made, the MIN_S algorithm computes the costs of the edges to the next layer; the cost of the edge on P_i is the cost of serving the request by A_i, as if A_i had been continually simulated from the beginning.

Then, the MIN_S algorithm applies the LGT algorithm in order to decide how to serve the request. If the LGT algorithm continues with the current path P_i, then MIN_S continues to simulate the current algorithm, A_i. On the other hand, if the LGT algorithm backtracks and moves to a vertex v (in the next layer) via another path, P_j, then the MIN_S algorithm switches to the configuration corresponding to v, and A_j becomes the current algorithm. Since we assumed that the underlying problem is a metrical task system, the triangle inequality holds for the cost of switching between configurations; thus the cost of MIN_S is bounded by the cost of LGT. Clearly then, a competitive LGT algorithm yields a MIN_S algorithm with the same competitive ratio, or better.

Conversely, one can easily use a MIN_S algorithm to construct an LGT algorithm with the same competitive ratio: Let the metrical task system P be the disjoint-paths layered graph traversal, where the states correspond to vertices in the graph, with the transition cost between states equal to the total distance in the graph. (It is readily seen that P is well defined.) Let $A_1, A_2, \ldots, A_m$ be the m algorithms that correspond to sticking to path P_i and let $a_1, \ldots, a_m$ be the waste factors. Clearly, the LGT algorithm that follows MIN_S in the obvious manner has the same competitive ratio.

3 The Guess Game

In this section we define and analyze a certain two player zero-sum game, called the *Guess Game*. Later we will use this analysis to derive upper and lower bounds for the disjoint-paths layered graph traversal problem.

One participant is called the *player* and the other is called the *adversary*. Both start with the same set of cards $T = [m] = \{1, \ldots, m\}$. The value of card i is $a_i > 0$.

The game starts with the adversary putting all his cards face-down on the table in a certain order, that he will be unable to change during the game. Then the player chooses one of her cards and puts it face-down on the table. We call this the *hidden* card. The adversary then turns up the first of his cards and the player has to match it. If the card matches the hidden card (a hit) then the adversary wins the value of the card, the matched pair is discarded, and the player must pick a new card face down. If not (a miss), then the player matches the adversary's card with a card from her hand and the matched pair is discarded without further ado.Hence, there are m rounds. The value of the game is the sum of the values of the cards that the adversary wins.

Observe that the player pays a_i if and only if she hides card i before she hides any card that comes after i in the adversary's order. In particular the player always pays for the last card in the adversary's order.

Let's assume that that the player selects her algorithm first, and that the adversary is aware of the selection made. If the player's algorithm is deterministic, then the adversary's best strategy is obvious: he chooses the order of his guesses to be the same order as the hidden cards of the player and thus he wins at every round. Hence, the value of the game is exactly $\sum_{1\le i\le m} a_i$.

In the randomized case the situation is more complicated. The player can choose her hidden cards according to distributions that might depend on the history of the game. On the other hand, basic game theory implies that, given the probability distribution on the player strategies, there is a deterministic strategy for the adversary, that is, a fixed order of guesses, that maximizes his profit.

In order to analyze the value of the game we define two other models for players. A *strong* player is a player which, after each miss, is allowed to replace the hidden card by a card which is still in her hand. This, of course, can only help the player and does not increase her expected cost with respect to a standard player. A *weak* player is one that chooses the order of her hidden cards in advance (using random bits) and is not allowed to change this order later in the game. More precisely, the weak player chooses an order for her cards at the beginning of the game and then, whenever there is a hit, she replaces the hidden card by the lowest ordered card which has not been discarded yet. Clearly the expected cost for a weak player is no lower than the expected cost for a standard player.

Let $f(T)$ be defined by equation (1.1). Our main result in this section is

THEOREM 3.1. *The value of the game with a set of cards T is at least $f(T)$, even against a strong player, and is at most $f(T)$ even against a weak player. Thus the game value is exactly $f(T)$ for all three types of players.*

Proof. We start with the lower bound and assume a strong player. Let $g(T)$ be the value of the game. We have to show that $g(T) \ge f(T)$ for any set T. We use induction on the size of T. If $T = \{i\}$, then $g(T) = a_i = f(T)$ and we are done. For the general case, let p_j be the probability that the player chooses card j as her first hidden card. Now, if the adversary chooses card i to be his first guess, and then chooses the best order for the remaining cards as if the game started with $T \setminus \{i\}$, he can clearly guarantee, even against a strong player, an expected cost of at least $p_i a_i + g(T \setminus \{i\})$. The adversary can choose the i which maximizes this expression. That implies that for all i

$$g(T) \ge p_i a_i + g(T \setminus \{i\}),$$

or

$$\frac{g(T) - g(T \setminus \{i\})}{a_i} \ge p_i.$$

But $\sum_i p_i = 1$, and therefore

$$\sum_i \frac{g(T) - g(T \setminus \{i\})}{a_i} \ge 1$$

or

$$g(T) \ge \frac{1 + \sum_{i\in T} g(T \setminus \{i\})/a_i}{\sum_{i\in T} 1/a_i}$$

Thus $g(T) \ge f(T)$.

We now turn to the upper bound and assume a weak player. Again the proof is by induction on the size of T. The case $T = \{i\}$ is trivial. For the general case, recall that a weak player hides her cards in a fixed order. Assume that the player constructs her order as follows: Among all cards she picks a card with probability inversely proportional to its value. Let the card so chosen be the *last* card in her order. From the remaining cards she picks again a card with probability inversely proportional to its value. Let it be the next-to-last card in her order. And so on. (That is, if after k choices the set of remaining cards is T and $i \in T$, the probability that i becomes the $m-k$ card in the order is $(1/a_i)/\sum_{j\in T} 1/a_j$.)[2]

Let $h(T)$ be the value of the game when the adversary knows that the player has chosen this particular strategy. Let j be the card chosen by the adversary to be last in his order. Let i be the last card of the player. Note that j is fixed, but i is a random variable.

- If $i = j$, an event whose probability is proportional to $1/a_j$, then the player has to pay a_j in the last round. Furthermore, the distribution used by the weak player with respect to the set of remaining cards (that is, $T \setminus \{j\}$) is exactly the same as if she started the game with the set $T \setminus \{j\}$. Hence in this case, the player's expected cost is at most $a_j + h(T\setminus\{j\})$ even if the adversary plays optimally on the remaining cards.
- If $i \ne j$, the player will never have to pay a_i and again her distribution on the remaining cards is exactly as if she had started the game with $T\setminus\{i\}$, so her cost is at most $h(T \setminus \{j\})$.

[2]Note that this strategy is not the same as choosing the sequence from first to last with probabilities proportional to a_i.

This implies that

$$h(T) \leq \frac{1/a_j}{\sum_{i\in T} 1/a_i}(a_j + h(T \setminus \{a_j\})) + \frac{\sum_{i\in T\setminus\{j\}} h(T\setminus\{i\})/a_i}{\sum_{i\in T} 1/a_i}$$

or

$$h(T) \leq \frac{1+\sum_{i\in T} h(T\setminus\{i\})/a_i}{\sum_{i\in T} 1/a_i}$$

That is, $h(T) \leq f(T)$.

We conclude that $h(T) = f(T) = g(T)$ and thus the value of the game is exactly $f(T)$ for all three types of players. □

3.1 Properties of $f(T)$.

In this subsection we discuss some of the interesting properties of $f(T)$.

Let's return to the weak player's strategy as described in Theorem 3.1. Let $P(i, R)$ for $i \in R \subset T$ be the probability that the player chooses card i the last among the cards in R (which means that, in the player's hiding order, card i will be the first among the cards in R.) We claim that $P(i, R)$ does not depend on the values of the cards in $T \setminus R$. Indeed, call the cards in R, *red*. We can think that when the player builds her order, she first decides, with suitable probability, *whether* to pick a red card from the remaining cards, and if so, she then decides, with suitable probability, *which* red card to pick. Clearly the order among the red cards depends only on the values of the red cards.

Let $\pi_1, \pi_2, \ldots, \pi_m$ be the adversary's order. As we have already observed, for any strategy, the player pays a_i if and only if she hides card i before she hides any card that comes after i in the adversary's order. That implies that the probability that the weak player pays a_{π_i} is exactly, $P(\pi_i, \{\pi_i, \pi_{i+1}, \ldots, \pi_m\})$.

But the proof of Theorem 3.1 implies that the weak player's strategy as described is optimal, and therefore game theoretical considerations imply that the order chosen by the adversary is irrelevant – the expected value of the game is the same. It follows that

$$\text{(3.3)}\quad f([m]) = \sum_{1\leq i\leq m} a_{\pi_i} P(\pi_i, \{\pi_i, \pi_{i+1}, \ldots, \pi_m\})$$

for any permutation π! In particular,

$$\text{(3.4)}\quad f([m]) = \sum_{1\leq i\leq m} a_i P(i, \{i, i+1, \ldots, m\}).$$

In this form, it is rather hard to see that $f([m])$ is symmetric in the a_i's, since the i'th term in the sum depends only on $a_i, a_{i+1}, \ldots, a_m$. We also don't know of any direct proof that shows that (3.4) is a solution of (1.1).

Unfortunately, the alternate expression is not computationally easier, since $P(i, R)$ does not seem to have a simple closed form. It can be computed with the formula

$$\text{(3.5)}\quad P(i, R) = \frac{\sum_{j\in R\setminus\{i\}} P(i, R\setminus\{j\})/a_j}{\sum_{j\in R} 1/a_j}$$

Similar considerations lead to

THEOREM 3.2. *Let $T = [m]$. Without loss of generality assume that $a_1 \leq a_2 \leq \cdots \leq a_m$. Then*

$$\sum_{1\leq i\leq m} \frac{a_i}{i} \leq \sum_{1\leq i\leq m} \frac{a_i^2}{a_1+\cdots+a_i} \leq f(T)$$

and

$$f(T) \leq \sum_{1\leq i\leq m} \frac{a_i^2}{a_i+\cdots+a_m} \leq \sum_{1\leq i\leq m} \frac{a_i}{m+1-i}.$$

Proof. As above we consider the weak player's strategy. It suffices to show that if $a_1 \leq a_2 \leq \cdots \leq a_m$ then

$$\text{(3.6)}\quad P(1, [m]) \leq \frac{a_1}{a_1+\cdots+a_m},$$

and

$$\text{(3.7)}\quad P(m, [m]) \geq \frac{a_m}{a_1+\cdots+a_m}.$$

The proof is by induction on m. Let $S = a_1 + \cdots + a_m$. The base case is trivial. For the general case, by the definition of the weak player's strategy and the induction hypothesis, we have

$$\text{(3.8)}\quad \begin{aligned} P(1, [m]) &\leq \sum_{j>1} \frac{1}{a_j}\frac{a_1}{S-a_j} \Big/ \sum_j \frac{1}{a_j} \\ &= \frac{a_1}{\sum_j 1/a_j} \sum_{j>1} \frac{1}{a_j}\frac{1}{S-a_j} \end{aligned}$$

Observe that

$$\frac{1}{a_j}\frac{1}{S-a_j} = \frac{1}{S}\left(\frac{1}{a_j} + \frac{1}{S-a_j}\right).$$

Hence (3.8) becomes

$$P(1, [m]) \leq \frac{a_1}{S}\sum_{j>1}\left(\frac{1}{a_j} + \frac{1}{S-a_j}\right) \Big/ \sum_j \frac{1}{a_j},$$

for which it suffices to show that

$$\sum_{j>1} \frac{1}{S-a_j} \le \frac{1}{a_1}.$$

Similarly, proving equation (3.7) reduces to proving that

$$\sum_{j<m} \frac{1}{S-a_j} \ge \frac{1}{a_m}.$$

The last two inequalities follow from

$$\sum_{j>1} \frac{1}{S-a_j} \le \sum_{j>1} \frac{1}{(m-1)a_1} = \frac{1}{a_1},$$

and

$$\sum_{j<m} \frac{1}{S-a_j} \ge \sum_{j<m} \frac{1}{(m-1)a_m} = \frac{1}{a_m}.$$

Now using equation (3.6) (resp. (3.7)) in equation (3.3) and the permutation $\pi_i = m-i+1$ (resp. $\pi_i = i$) completes the proof. □

4 The lower bound

Let r be an arbitrary positive real number. Given $a_1, \ldots, a_m$, we show that an adversary can construct a disjoint-paths layered graph such that the cost of the off-line (modified) LGT algorithm is r while the cost of any on-line LGT is at least $rv([m])$, where $v([m])$ is the value of the Guess game on m cards with values $a_1, \ldots, a_m$. (If the on-line LGT algorithm is deterministic (resp. randomized) then so is the player; and the value of $v([m])$ changes accordingly: $v([m]) = \sum_{1\le i\le m} a_i$ in the deterministic case and $v([m]) = f([m])$ in the randomized case.)

The graph consists of m paths and $m+1$ layers. Each path starts with an edge with finite positive length, followed by a number of zero length edges, followed by a (practically) infinite length edge, except for the path to the target, which does not contain the infinite edge. Path P_i starts with an edge with length ra_i and has waste factor a_i. Each path has its infinite edge starting on a different layer, and for $j = 1, \ldots, m-1$, every layer L_j has an infinite edge out.

Thus, the on-line algorithm has no reason to visit the same path twice, and whenever it paid the first edge on the path, it can be presumed that it will not backtrack before reaching the infinite edge, or the target, since it costs nothing to advance and return on the zero length edges.

Path P_i corresponds to card i in the game. The on-line algorithm starting path i corresponds to the player hiding card i. An infinite edge on P_i between layer j and $j+1$ corresponds to the adversary guessing card i at round j in the game. With these correspondences, it can be easily verified (see the example below) that a (randomized) strategy for the LGT algorithm immediately translates into a strategy for the (randomized) standard player in the Guess game.

Given the player's strategy, the adversary starts by choosing an order on the paths corresponding to his optimal order of guesses in the Guess game. Let this order be $\pi_1, \pi_2, \ldots, \pi_m$. The adversary completes the construction as follows:

- The target is on path π_m at layer m.
- For $i = 1, \ldots, m-1$ the path π_i gets its infinite edge between layers i and $i+1$.

For instance if the algorithm starts on path π_3 then it pays ra_{π_3} but will not pay the first edges on π_1 and π_2. This corresponds exactly to the player in the Guess game that hides π_3 at the first turn, and hence pays a_{π_3} but does not pay a_{π_1} or a_{π_2}.

Clearly the cost of the LGT algorithm is at least r times greater than the cost of the player. ("At least" because the LGT algorithm also pays on the way back to source.) Hence we conclude that the cost of the on-line LGT algorithm is at least $rv([m])$ while the cost of the off-line algorithm is only r. (The path to the target has length ra_m and waste factor a_m.) This concludes the proof of the lower bound.

5 The upper bound

The proof below makes the assumption that the algorithms $A_1, \ldots, A_m$ are deterministic. The proof for randomized algorithms is similar, but requires many technicalities which obscure the essential ideas. We leave it for the full paper.

We are given a disjoint-paths layered graph traversal problem: the graph consists of m paths $P_1, \ldots, P_m$ with waste factors $a_1, \ldots, a_m$. We show how to construct an LGT algorithm using a strategy for the Guess game.

Let $l_{i,j}$ be the distance from the source to layer j on path P_i. Let $s_j = \min_i l_{i,j}/a_i$, that is, s_j is the minimum cost that the off-line algorithm must pay to get to layer j. Let j' be the minimum j such that $s_j > 0$. Let $s = s_{j'}$.

We now partition the layers into *strata*. All layers j such that

$$s2^{k-1} \le s_j < s2^k,$$

belong to stratum k. All layers j with $j < j'$ belong to stratum 0.

Consider a layer j in stratum k. If $l_{i,j}/a_i > s2^k$ call path i *blocked at* j. Notice that on each layer there must be some path which is not blocked. (If all the paths have $l_{i,j}/a_i > s2^k$ then we just started a new stratum, and the "blocked" notion is redefined.) For stratum 0 we call a path blocked if $l_{i,j}/a_j > 0$.

Now we are ready to describe the on-line LGT algorithm. For stratum 0, the algorithm follows a 0-length path until it blocks, then switches arbitrarily to another 0-length path, and so on, until all paths have strictly positive lengths and stratum 1 starts. Notice that in general, once the algorithm has reached layer $j-1$, it can compute s_j at no cost.

Once it gets to the first layer of stratum k, the on-line algorithm gets to the first layer of the next non-empty stratum k' this way: it follows a path until it blocks (with respect to stratum k), then it returns to the source and follows another path not yet blocked, and so on, until it arrives on the first layer of stratum k'. The crux of the algorithm is how to choose the next path to try.

The idea is that on the first layer of stratum $k > 0$, the algorithm starts playing a Guess game. As in the lower bound proof, path P_i corresponds to card i in the game, the on-line algorithm trying path i corresponds to the player hiding card i, and a blocked path P_i corresponds to the adversary guessing card i at a certain round in the game.

The difference is that now the adversary might guess (and miss) some cards even before any card is hidden – this corresponds to paths that are already blocked with respect to stratum k on the first layer of the stratum, and the adversary might guess several cards at once, against a single hidden card – this corresponds to paths that block on the same layer. Of course, both these maneuvers work to the advantage of the player.

Notice that card i on stratum k costs at most $s2^k a_i$. Taking into account backtracking, the total cost of the on-line algorithm on stratum k is bounded by $2s2^k v([m])$ where $v([m])$ is the value of the Guess game on m cards with values $a_1, \ldots, a_m$.

Assume that the target belongs to stratum t. Then the total cost of the off-line algorithm is at least $s2^{t-1}$ while the total cost of the on-line algorithm is at most

$$2s(1+2+\ldots+2^t)v([m]) < s2^{t+2}v([m]).$$

Hence the competitive ratio is at most $8v([m])$: that is, $8\sum_{1\leq i\leq m} a_i$ for the deterministic case, and $8f([m])$ for the randomized case.

6 Application to the k server problem

The k-server algorithm of [5] is based on recursive calls to the MIN$_S$ operation with i^2 algorithms whose competitive ratio can be divided into i groups, each of size i. The competitive ratios of the algorithms in the same group are about the same but the ratios differ greatly among groups. More precisely, the sum of the competitive ratios of all the algorithms is dominated by the sum in one group. Thus the average competitive ratio is $\Theta(i)$ times smaller than the maximum one. The MIN$_S$ originally used in [5] has competitive ratio $O(m \cdot \max_i\{a_i\})$, while our algorithm has competitive ratio $O(\sum_{1\leq i\leq m} a_i)$. Hence, our algorithm saves a $\Theta(i)$ factor in each recursive call and this results in a $k!/2^{O(k)}$ overall savings factor. Unfortunately the competitive ratio of the modified algorithm is still exponential in k, namely $O((k!)^2 2^{O(k)})$.

A Combining off-line algorithms.

Let S be a set of k off-line algorithms such that for each input at least one of algorithms runs quickly. What is the fastest way to combine the execution of the algorithms in S to solve a particular input? Can we find an algorithm which combines the elements of S which, for every input, achieves performance within some constant factor of the fastest algorithm for that input?

Again we are interested in a combining procedure that has no specific knowledge of the problem domain and input. We consider two models: in the first one there is no a priori bound on the running time of the algorithms; in other words the only way to determine the running time of a particular algorithm on a given input is to run it until it terminates. In the second model, the running time of each algorithm A_i is known to be either exactly a_i or infinite; this corresponds to having a known competitive ratio, or performance guarantee.

A.1 No performance guarantee. The standard solution to this problem is the Round Robin (RR) algorithm which achieves a performance ratio of precisely k in the worst case. It works by executing individual instructions from each of the algorithm in turn until one of them terminates. Thus, if the fastest algorithm $A \in S$ costs n steps, RR will cost between $k(n-1)+1$

and kn, depending on where A falls in the ordering of S. If the ordering is deterministic, the adversary can chose an input such that the last algorithm in the order is the best, leading to a competitive ratio of exactly k. It is easy to see that RR is optimal among deterministic combining algorithms.

Can randomization help reduce the expected ratio? Consider, for example, the algorithm that first randomly sorts the algorithms in S, and then applies round robin. The expected cost on an input with least cost n is then $k(n-1)+(k+1)/2 = kn-(k-1)/2$, yielding a competitive ratio approaching k as n becomes large. This algorithm fails to improve the competitive ratio in the worst case.

We now show that we cannot hope to do better by showing that no algorithm can achieve a ratio better than k. To prove this, we apply a variation of Yao's theorem to the competitive ratios under consideration (not to the costs themselves!), which allows us to replace randomness in the algorithm with randomness in the input. We will choose a distribution on the identity of the fastest algorithm among the k algorithms in S and its termination time. Let n be a parameter to be chosen later and suppose that the other algorithms have infinite cost on the inputs for which they are not fastest; it suffices for these costs to be at least kn.

Now, let the probability that the A_j is the fastest algorithm, and that its termination time is i (where $1 \le i \le n$), be

$$\frac{1}{k} \cdot \frac{i}{1+\ldots+n} = \frac{2i}{kn(n+1)}$$

This defines a probability distribution. Take any combining algorithm C for this distribution. We will show that it achieves a ratio no better than $(kn+1)/(n+1)$. For large enough values of n, this approaches k.

Why can C do no better than the ratio above? Since C is deterministic, and has no specific knowledge of the problem domain and input, it has a fixed order in which it simulates the steps of the algorithms. (For instance step 1-10 of algorithm A_7, followed by steps $1-15$ of algorithm A_5, and so on.) C stops as soon as one algorithm finishes. In the worst case C has to simulate kn steps.

Consider step t of C. At that step, suppose C simulates step i of algorithm j. With probability $2i/(kn(n+1))$, this will be the terminating step, leading to a cost ratio of t/i. This contributes $2t/(kn(n+1))$ to the expected ratio, independently of i and j. Thus for every order, and hence for every algorithm, the expected ratio is

$$\sum_{1\le t\le kn} \frac{2t}{kn(n+1)} = \frac{kn+1}{n+1}.$$

A.2 A priori performance guarantee. Let S be a set of n off-line algorithms such that for each input at least one of algorithms runs quickly. Suppose that the running time of each algorithm i is known to be either exactly a_i or infinite. Again we are interested in a procedure which combines the algorithms such that for every input, it achieves a performance within some constant factor of the fastest algorithm.

First, observe that our desired algorithm need never interleave the executions of different algorithms. Since no information about the running time of algorithm i is gained until step a_i, we can convert any algorithm for this problem into one which runs the algorithms in the order in which their decisive steps are executed.

Therefore, our algorithm is determined by its ordering of the algorithms from S. If the ordering is deterministic, the worst case cost is $s = \sum_{1\le i\le n} a_i$, achieved when the input is solved only by the last algorithm tried.

In this case, randomization does help. The exact complexity for the randomized case is

$$\sum_{1\le i\le j\le n} a_i a_j \Big/ \sum_i a_i$$

Lower bound: We use Yao's theorem. The adversary assigns probability a_j/s to the outcome that j is the correct algorithm. Consider a deterministic combining algorithm C that chooses an execution order $p_1, p_2, \ldots, p_n$. If p_j was the correct algorithm, then C incurs cost $\sum_{1\le i\le j} a_{p_i}$. Thus, the total expected cost is

$$\sum_{1\le j\le n} \frac{a_{p_j}}{s} \sum_{1\le i\le j} a_{p_i} = \sum_{1\le i\le j\le n} a_i a_j \Big/ \sum_i a_i.$$

Upper bound: Arrange the algorithms in some order e.g. $1, 2, \ldots, n$. With probability a_j/s start with algorithm j, then $j+1$, and so on in cyclic order. Let l be the index of the correct algorithm. If the algorithm starts with j its cost is $\sum_{j\preceq i\preceq l} a_i$ where $\sum_{j\preceq i\preceq l}$ denotes a *cyclic sum*. A cyclic sum is the same as a regular sum when $j \le l$, but if $j > l$, then the sum is on the indices i that satisfy $j \le i \le n$ and $1 \le i \le j$. Then the expected cost is

$$\sum_{1\le j\le n} \frac{a_j}{s} \sum_{j\preceq i\preceq l} a_i = \sum_{1\le i\le j\le n} a_i a_j \Big/ \sum_i a_i.$$

References

[1] R. Baeza-Yates, J. Culberson and G. Rawlins, "Searching in the plane," to appear in *Information and Computation.*

[2] A. Borodin, N. Linial, and M. Saks. "An Optimal On-line Algorithm for Metrical Task Systems" *Proceedings of the 19th Annual ACM Symposium on Theory of Computing,* 1987, pp. 373–382.

[3] A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, and N. Young. "Competitive paging algorithms," *Journal of Algorithms*, 12(1991), pp. 685-699.

[4] A. Fiat, D. Foster, H. Karloff, Y. Rabani, Y. Ravid and S. Vishwanathan, "Competitive algorithms for layered graph traversal," *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, 1991, pp. 288-297.

[5] A. Fiat, Y. Rabani and Y. Ravid, "Competitive k-server algorithms," *Proceedings of the 31st IEEE Symposium on Foundations of Computer Science*, 1990, pp. 454-463.

[6] C. Papadimitriou and M. Yanakakis, "Shortest paths without a map," *Proceedings of the 16th ICALP*, 1989, pp. 610-620.

[7] H. Ramesh, "On traversing layered graphs on-line," *This proceedings*, 1992.

[8] D. Sleator and R. Tarjan, "Amortized efficiency of list update and paging rules," *Communications of the ACM*, 23(1985), pp. 202–208.