The Server Problem and On-line Games

Marek Chrobak Lawrence L. Larmore
Department of Computer Science
University of California
Riverside, CA 92521

June 4, 2011

Contents

1 Introduction

1.1 On-line Problems

Classical theory of algorithms deals with computational problems in which an algorithm is assumed to have complete knowledge of the input data. In some applications, however, this setting is not realistic. Sometimes only partial information about data is available, and the algorithm is supposed to compute, or at least approximate, the desired function based only on this partial information.

An important class of problems involving partial information are *on-line problems*. In those problems data is supplied to the algorithm in units, one unit at a time. The algorithm outputs a string of outputs: after seeing t units of input, it needs to print the t^{th} unit of output. Thus the algorithm makes a decision based only on partial information about the whole input string, namely the part of the input string that has been read so far. How good the decision of the algorithm at step t is may depend on the future inputs, inputs that the algorithm has not yet seen.

An *optimization* problem is one where the goal is to find that output of minimum "cost" (or, perhaps, maximum "return") given the inputs. There is a tremendous literature on off-line optimization problems.

In this talk, we deal exclusively with *on-line optimization problems*. Since we will not deal with any other kind, we will say "on-line problem" when we mean "on-line optimization problem."

1.2 Competitiveness

In on-line optimization problems it is, in general, impossible to compute the optimal solution. For example, consider a simple "bit-matching" game: the input consists of one bit, the output also consists of one bit, and the "cost" is 1 if the output matches the input, 2 if it does not. The optimum solution is for the input to match the output, achieving the minimum cost of 1. But in the on-line case, when that output must be given *before* the input is read, it is impossible to achieve this optimal cost, in fact, the lowest cost that can be guaranteed is 2. We need some way to measure the performance of on-line algorithms in which this cost of 2 is considered optimal, since no on-line algorithm can do better for this problem.

To handle this situation, the concept of *competitiveness* was defined by Manasse, McGeoch, and Sleator [15, 14]. The same idea was introduced earlier by Sleator and Tarjan in their paper on maintaining linear lists [18].

Suppose that $\mathbf{P}$ is an on-line problem, and $\mathcal{A}$ is an on-line algorithm for $\mathbf{P}$. Let $c \geq 1$ be a constant. We say that an on-line algorithm $\mathcal{A}$ for $\mathbf{P}$ is *c-competitive* if, for any instance I of problem $\mathbf{P}$,

$$cost_{\mathcal{A}}(I) \leq c \cdot cost_{opt}(I) + b,$$

where b is a constant that does not depend on the input string I.[1] Thus algorithm $\mathcal{A}$, asymptotically, pays at most c times the optimal cost, for a given input string. This general notion of competitiveness will be defined more precisely in the section about on-line games.

For example, any algorithm for the bit-matching game described above is 2-competitive, with the additive constant $b = 0$.

We say that a given on-line problem $\mathbf{P}$ is *c-competitive* if there exists a c-competitive algorithm for $\mathbf{P}$, and we say that it is *no better than c-competitive* if there exists no *c′-competitive* algorithm for $\mathbf{P}$ for any $c' < c$.

For example, if $\mathbf{P}$ consists of repeated plays of the bit-matching game, $\mathbf{P}$ is 2-competitive and is no better than 2-competitive.

Competitiveness is commonly used as a measure of performance in the literature of on-line problems. An on-line algorithm $\mathcal{A}$ is then said to be "optimal" for $\mathbf{P}$ if $\mathcal{A}$ is c-competitive and $\mathbf{P}$ is no better than c-competitive.

An on-line problem can be viewed as a game against an "adversary," which can be thought of as an opponent who chooses the inputs. If $\mathcal{A}$ is an on-line algorithm, we will assume that

[1] In some examples, for instance the server problem, it is convenient to allow the constant to depend on that part of the input, say I^0, which is read before any output is written. Thus, we would say that $\mathcal{A}$ is c-competitive if $cost_{\mathcal{A}}(I) \leq c \cdot cost_{opt}(I) + b(I^0)$. This formulation is clearly equivalent, since I^0 could be considered to be part of $\mathbf{P}$ instead of part of the instance I.

the adversary can examine the code for $\mathcal{A}$ and choose the input stream I in the worst possible way.

1.3 Randomized Algorithms

A *randomized algorithm* $\mathcal{A}$ can be considered to be a random variable over the set of all deterministic algorithms.

A randomized algorithm $\mathcal{A}$ is defined to be *c-competitive* if, for any problem instance I,

$$E[cost_{\mathcal{A}}(I)] \leq c \cdot cost_{opt}(I) + b,$$

for some additive constant b, where $E[cost_{\mathcal{A}}(I)]$ denotes the expected cost of $\mathcal{A}$ given input I.

Thus a randomized algorithm is just a mixed strategy in a game against a so-called *oblivious adversary*, who can examine the code for $\mathcal{A}$, but who cannot predict the outcome of the random choices within $\mathcal{A}$ and cannot see $\mathcal{A}$'s outputs.

1.4 The Overview of the Paper

In this section we will review the results presented in this paper.

Section 2 deals with the k-server problem. First we review previous work on the problem and introduce the notion of convex hulls of metric spaces that was defined in [5]. Then we propose a new algorithm for k servers that we call *Equipoise*. We prove that Equipoise is 2-competitive for two servers and 11-competitive for three servers. This is by far the smallest known competitive constant for $k = 3$.

In Section 2 we also present another algorithm for k servers that we call the *Work Function Algorithm* (WFA). Based on the overhelming empirical evidence, we conjecture that this algorithm is optimal, *i.e.*, k-competitive. We give a complete proof of this fact for $k = 2$, and later we outline our current method of attack against the general case. We believe that the concept of the WFA generalizes to a wide collection of other on-line problems (see Section 5).

In Section 3 we introduce a general model for on-line problems that we refer to as *on-line games*. Such an on-line game is described by a set of states Q, a set of request $\mathcal{R}$ and a transition function f. The value $f(x, r, y)$ is the cost of moving from state x to state y on request r. All on-line problems that we have dealt with so far can be expressed as on-line games.

Our purpose has been to describe in a formal mathematical setting some phenomena that have been observed by us and other researchers in the work on on-line problems. First we consider games where negative costs are allowed, and investigate so-called *finite-cost* algorithms,

i.e., algorithms that never pay more than a certain constant (independent of the request sequence). We prove that the existence of such an algorithm is equivalent to the existence of a *potential function*. The potential method has been used extensively in proofs of competitiveness of various on-line problems. Our result explains why this technique has proven so useful, since it shows that, in fact, constructing a potential function, whether explicitly or not, is unavoidable.

Later we show that such potential functions can be computed using a certain *update* operator. We used this method extensively in our computer programs for testing the k-server conjecture, and for finding (or showing non-existence of) competitive algorithms for other on-line problems.

Next, we consider games with non-negative costs. We show that the problem of finding a c-competitive algorithm for such a game $\mathcal{G}$ can be reduced to finding a *memoryless finite-cost* algorithm in another game $\mathcal{G}'$, in which negative costs are allowed, that can be constructed from $\mathcal{G}$. This is an extension the analogous observation from [15] about the server problem. Finally, we also consider briefly the randomized case. We believe that a reduction similar to the one outlined above is possible in this case, too. However, we have not yet completed all the proofs.

In Section 4 we introduce another formalization of on-line problems called *Metrical Service Systems*. In an MSS we are given a metric space M and a set of possible requests $\mathcal{R}$, where each request $r \in \mathcal{R}$ is a subset of M. We have one server that can occupy and move between points of M. Given a request r, in order to serve r, our algorithm needs to move the server to one of the points in r.

The notion of MSS's is less general than on-line games. On the other hand, it is easy to see that the server problem can be modeled as a MSS. By choosing the metric space M and the set of requests $\mathcal{R}$ one can obtain a number of interesting and natural problems. For example, let M be the uniform space and let $\mathcal{R}$ contain the sets of cardinality at most m. For this problem we give an m-competitive deterministic algorithm and an H_m-competitive randomized algorithm, both optimal. If M is arbitrary and $m = 2$ we obtain the *2-point request problem*. For this problem we give a 9-competitive deterministic algorithm and a c-competitive randomized algorithm where $c \approx 4.59112$. We believe that both constants are optimal, but have not been able to prove it so far.

2 The Server Problem

The k-server problem introduced by Manasse, McGeoch, and Sleator [15, 14] is a special case of an on-line optimization problem.

A given problem instance consists of a positive integer k, and a metric space M. The data

consists of an initial configuration $S^0 \in \Lambda^k M$, (where $\Lambda^k X$ = the set of all multisets of order k within a set X) and a sequence of *requests* $r^1, r^2, \ldots r^m \in M$. We think of S^0 as the initial positions of k movable *servers*. After each request $r^t \in M$ we must move one server to r^t in order to "serve" the request. We are also free to move the other servers. Cost is defined as the total distance moved by all servers.

More formally, the output consists of a sequence $S^1, S^2, \ldots S^m \in \Lambda^k M$, such that $r^t \in S^t$, and cost is measured as $\sum_{t=1}^{m} S^{t-1}S^t$, where, for $S, S' \in \Lambda^k M$, SS' is the "minimum matching" distance between the two multisets, defined in 2.2.

Manasse, McGeoch, and Sleator [15] proved that the k-server problem is no better than k-competitive for any metric space with at least $k+1$ points. The problem of whether there is a general on-line algorithm with competitiveness k remains open.

For $k = 2$, the problem was solved by Manasse, McGeoch, and Sleator [15] who gave a 2-competitive algorithm for 2 servers. Another optimal solution was presented in [5]. A great deal of effort has been made to improve the time and space efficiency of 2-server algorithms [7, 13], including randomized algorithms [4, 17].

For some special cases, k-competitive algorithms have been found. Chrobak *et al.* [3] gave a k-competitive algorithm for the line and the weighted cache problem. This was generalized later to trees in [8]. Coppersmith *et al.* [9] discovered a randomized algorithm with competitiveness constant k that works for a class of metric spaces called *resistive*.

Although, for arbitrary k, no k-competitive algorithm is known, there are already some partial results in this direction. Fiat *et al.* [11] have presented a competitive algorithm for k servers whose constant is an exponential function of k. Grove [12] has proven that the harmonic algorithm is competitive for any k (also with an exponential constant). His result yields another competitive deterministic algorithm via the technique developed by [1].

Randomization is a powerful tool in the design of competitive algorithms. The uncertainty about the future data can be partially compensated for by allowing an algorithm to make probabilistic choices. For example, it is known that there is an H_k-competitive randomized algorithm in the uniform space [10, 16], far better than the lower bound of k for deterministic algorithms.

2.1 Convex Hulls

In this subsection, we let M be an arbitrary metric space. We let xy denote the distance in M between two points $x, y \in M$.

We define two larger metric spaces $\mathbf{Cl}(M)$, the *closure* of M, and $\mathbf{CH}(M)$, the *convex hull* of M, such that $M \subseteq \mathbf{CH}(M) \subseteq \mathbf{Cl}(M)$.

Let $\Delta(a,b,c)$ be the predicate, defined for all real a,b,c, which is true if and only if there could exist a triangle whose sides have lengths a,b,c. Thus, $\Delta(a,b,c)$ if and only if $a \leq b+c$, $b \leq c+a$, and $c \leq a+b$. We define $\mathbf{Cl}(M)$ to be the set of all real-valued functions u on M for which

$$(\forall x, y \in M) \ : \ \Delta\left(u(x), u(y), xy\right)$$

$\mathbf{Cl}(M)$ is a metric space under the $\mathcal{L}^\infty$ norm:

$$uv = \inf_{x \in M} |u(x) - v(x)|.$$

If $u, v \in \mathbf{Cl}(M)$, we say that $u \leq v$ if $u(x) \leq v(x)$ for all $x \in M$. This defines a partial order on $\mathbf{Cl}(M)$. We define $\mathbf{CH}(M)$ to be the set of all members of $\mathbf{Cl}(M)$ which are minimal under this partial order.

M embeds in $\mathbf{CH}(M)$ in a natural way. If $x \in M$, define $u_x \in \mathbf{Cl}(M)$ by $u_x(y) = xy$. It is easy to verify that u_x is minimal. By an abuse of notation, we shall identify x with u_x, and thus we can say $M \subseteq \mathbf{CH}(M)$.

Examples:

$\diamondsuit$ If $|M| = 2$, $\mathbf{CH}(M)$ is a line segment, with the two points of M at the ends.

$\diamondsuit$ If $|M| = 3$, $\mathbf{CH}(M)$ consists of three line segments joined at a point, to form a "Y" shape (See Figure 1). The points of M are at the ends of the arms.

$\diamondsuit$ If $|M| = 4$, $\mathbf{CH}(M)$ consists of a rectangle with the Manhattan metric, together with an line segment attached by one end to each corner. We call these segments "whiskers." The points of M are at the outer ends of the whiskers (See Figure 2).

$\diamondsuit$ If $|M| = 5$, $\mathbf{CH}(M)$ is a 2-dimensional polyhedron consisting of planar pieces (each with the Manhattan metric), attached to each other in various ways, and linear pieces (whiskers). Unlike for smaller k there is no one general shape of $\mathbf{CH}(M)$, but we have instead three general shapes.

$\diamondsuit$ The general case where $|M| = 6$ has not been worked out, but we conjecture that it consists of a three dimensional core, always some generalization of the semi-regular solid known as *rhombic dodecahedron*, together with 2-dimensional pieces (flaps) and 1-dimensional pieces (whiskers). The points of M are at the ends of the whiskers. In particular, if M consists of the six vertices of an octahedron, where the distance between antipodal points is 2 and the distances between other pairs of points is 1, $\mathbf{CH}(M)$ has the shape of the rhombic dodecahedron, with a special metric analogous to the Manhattan metric.

$\diamondsuit$ If M is finite, say $|M| = n$, let $\mathbf{R}^n$ have the sup-norm (*i.e.*, ℓ^∞) metric. Then $\mathbf{CH}(M) \subseteq \mathbf{R}^n$, and has the inherited metric, and consists of a connected finite union of compact pieces of various dimensions. No piece can have dimension greater than $\lfloor n/2 \rfloor$.

Lemma 2.1 *If M is any finite metric space, and $X \subseteq M$, then $\mathbf{CH}(X)$ can be isometrically embedded in $\mathbf{CH}(M)$ in such a way as to extend the embedding of X in M. Furthermore, if $M - X$ consists of a single point, that embedding is unique.*

More generally,

Lemma 2.2 *If M is any finite metric space, and $X \subseteq \mathbf{CH}(M)$ is also finite, then $\mathbf{CH}(X)$ can be isometrically embedded in $\mathbf{CH}(M)$ in such a way as to extend the embedding of X.*

Proofs of Lemmas 2.1 and 2.2 will be given in the journal version. We also expect to remove the finiteness condition, and give a more general discussion of convex hulls.

2.2 The Equipoise Algorithm

In this section, we introduce the *Equipoise Algorithm* for the k-server problem in an arbitrary metric space. The current status of this algorithm is as follows:

- For $k = 2$, the Equipoise Algorithm is provably 2-competitive, hence optimal, and is the same algorithm that was introduced in [5], but under a different name.
- For $k = 3$, the Equipoise Algorithm is provably 11-competitive.
- For $k > 3$, the Equipoise Algorithm has not been shown to be competitive at all.

Throughout the remainder of this section, we shall assume that M is finite. This is not a restriction, since the algorithm needs only consider the subspace consisting of all points which have been "mentioned" so far, *i.e.*, the initial positions of our servers and the set of points that have been requested up to the current step. A similar approach was used in [5].

Lemma 2.3 *If M' is a metric space, and if M embeds isometrically in M', and if there is a c-competitive algorithm for the k-server problem in M', then there is a c-competitive algorithm for the k-server problem in M.*

Proof: Servers just remember their virtual positions in M'. $\diamond$

Thus, since $M \subseteq \mathbf{CH}(M)$, we may assume that all servers are located in $\mathbf{CH}(M)$ when we describe the Equipoise algorithm.

If $X = \{x_1, x_2, \ldots, x_k\}$ and $Y = \{y_1, y_2, \ldots, y_k\}$ are multisets of the same cardinality in any metric space, we can define XY to be the minimum matching distance between the two multisets. Formally, that is defined to be

$$\min_{\sigma} \sum_{i=1}^{k} x_i y_{\sigma(i)}$$

where the minimum is over all permutations σ of the indices $\{1, \ldots, k\}$.

We now define the **Equipoise Algorithm:** Let $S = \{s_1, \ldots, s_k\} \subseteq \mathbf{CH}(M)$ be the current configuration of the server positions, and r the new request point. Let K be the multiset of current configurations and the new request point, *i.e.*, $K = \{r, s_1, \ldots, s_k\}$.

We define a weighted complete graph whose vertices are our current server positions, as follows: the weight of the edge $e_{ij} = (s_i, s_j)$ is

$$weight(s_i, s_j) = \frac{s_i s_j + s_i r + s_j r}{2}.$$

We call this weighted graph the *fictitious graph*. Let T be a minimal spanning tree of the fictitious graph, which consists of $k-1$ edges. If $e_{ij} \in T$, define $s'_{ij} \in \mathbf{CH}\{s_i, s_j\}$ to be that point whose distance to s_i is $\frac{s_i s_j + s_j r - s_i r}{2}$ and whose distance to s_j is $\frac{s_i s_j + s_i r - s_j r}{2}$. By Lemmas 2.1 and 2.2, we can write $s_{ij} \in \mathbf{CH}\{s_i, s_j\} \subseteq \mathbf{CH}(S) \subseteq \mathbf{CH}(K) \subseteq \mathbf{CH}(M)$. Let S' consist of r together with all s_{ij} for which $e_{ij} \in T$. Equipoise then chooses the minimum cost move of all servers from S to S'.

There is a certain arbitrariness in the definition of Equipoise. The embedding $\mathbf{CH}(K) \subseteq \mathbf{CH}(M)$ is not necessarily unique, but this does not seem to matter. The embedding $\mathbf{CH}(S) \subseteq \mathbf{CH}(K)$ is unique by Lemma 2.1, and for $k \leq 3$, the embedding $\mathbf{CH}\{s_i, s_j\} \subseteq \mathbf{CH}(S)$ is also unique. For $k \geq 4$, it is possible that the embedding $\mathbf{CH}\{s_i, s_j\} \subseteq \mathbf{CH}(S)$ will need to be carefully chosen to insure that Equipoise is competitive – this is an open question.

A proof of 2-competitiveness for $k = 2$ is given in [5]. It is based on a potential argument where the potential is defined to be

$$\Phi = s_1 s_2 + 2 \cdot SA,$$

where S and A are the configurations of our servers and the adversary servers, respectively.[2]

Since the publication of [5], we have developed a different kind of argument. We do not consider the positions of the adversary servers at all. Instead, we define a new quantity which we call *adversary pseudo-cost* for each of our moves. Initially our servers are on some position S^0. Let S be the current configuration of our servers, let r be the new request point and suppose that our algorithm moves one server so that the new configuration is S'. Our cost is SS' = the total distance moved by our servers, while the adversary pseudo-cost is defined to be

$$\nabla(S, r, S') = \min_{X \ni r} \left\{ SX - S'X \right\}.$$

Note that, without loss of generality, the minimum may be taken only over configurations $r \in X \subseteq \{r\} \cup S$, of which there are just k choices.

[2] This formula was proposed by Coppersmith *et al.* [9]

Lemma 2.4 *For any sequence of requests, the total adversary pseudo-cost for any algorithm for the k-server problem is less than or equal to the optimum cost for that request sequence.*

Proof: Let S^0 be the initial configuration of our servers. The adversary starts from the same configuration, *i.e.*, $S^0 = A^0$. Let $\varrho = r^1 \dots r^m$ be the request sequence. After request r^t our servers are in configuration S^t and the adversary servers are in configuration A^t. By the triangle inequality

$$\begin{aligned}\sum_{t=1}^{m} \nabla(S^{t-1}, r^t, S^t) &= \sum_{t=1}^{m} \min_{X \ni r^t} \{S^{t-1}X - S^tX\} \le \sum_{t=1}^{m} (S^{t-1}A^t - S^tA^t) \\ &= \sum_{t=1}^{m} (S^{t-1}A^t - S^{t-1}A^{t-1}) - S^mA^m \le \sum_{t=1}^{m} A^{t-1}A^t = cost_{opt}(\varrho)\end{aligned}$$

$\diamond$

Lemma 2.5 *If $k = 2, 3$, the adversary pseudo-cost for each move of Equipoise is equal to the distance moved by the one of our servers that moves to the request point, minus the distance moved by our other servers.*

Proof: Let first consider the case $k = 2$. $S = \{x, y\}$ is the current configuration of our servers, r is the new request point, and $S' = \{r, u\}$ is our server configuration after serving r.

The convex hull of $K = \{x, y, r\}$ consists of three "whiskers" meeting at a "branch" point, whose lengths are the three so-called *slack* variables, a, b, c. The point x is at the end of an whisker of length a, the point y is at the end of an whisker of length b, and r is at the end of an whisker of length c, where

$$\begin{aligned} xy &= a + b \\ xr &= a + c \\ yr &= b + c \end{aligned}$$

Without loss of generality, $a \le b$ (see Figure 1).

Equipoise moves one server from x to r, and the other server from y to the point u, on y's whisker, which is distance a from y (see Figure 3).

By definition, the adversary pseudo-cost is

$$\min\{xr - yu, yr - xu\} = c$$

which is also the distance moved by our server from x to r, minus the distance moved by our other server. This completes the proof for $k = 2$.

Now, let consider the case $k = 3$. We denote by $S = \{x, y, z\}$ the current configuration of our servers, r is the new request point, and $S' = \{r, u, v\}$ our server configuration after serving r.

Without loss of generality, we can assume that y matches with r in the maximal matching of $K = \{x, y, z, r\}$, *i.e.*,

$$yr + xz \ \geq \ xr + yz, \ zr + xy.$$

$\mathbf{CH}(K)$ has the appearance of a rectangle with four whiskers attached to the corners, and the members of K are at the ends of the whiskers. The rectangle has the Manhattan metric.

Let a, b, c, d, e, f be the six *slack parameters* of $\mathbf{CH}(K)$, which are defined to be non-negative reals satisfying the equations

$$\begin{aligned} xy &= a + b + e \\ xz &= a + c + e + f \\ yz &= b + c + f \\ xr &= a + d + f \\ yr &= b + d + e + f \\ zr &= c + d + e \end{aligned}$$

The lengths of the four whiskers are a, b, c, d, and the rectangle has dimensions $e \times f$ (See Figure 2). We recognize eight general cases, depending on relations among the slack variables. Taking into account the symmetry which exchanges $x \leftrightarrow z$, and which then also exchanges $a \leftrightarrow c$ and $e \leftrightarrow f$, we reduce this number to five cases.

We first compute the edge weights of the fictitious graph:

$$\begin{aligned} \mathit{weight}(x, y) &= a + b + d + e + f, \\ \mathit{weight}(x, z) &= a + c + d + e + f, \\ \mathit{weight}(y, z) &= b + c + d + e + f. \end{aligned}$$

Thus, the minimal spanning tree of the fictitious graph consists of the edges $(x, y), (x, z)$ if $a = \min\{a, b, c\}$ and $(x, y), (y, z)$ if $b = \min\{a, b, c\}$. (The case where c is the minimum is symmetric to the case where a is the minimum.)

Case I : $a \leq b$ and $a + f \leq c$. Then $uy = a$, and $vz = a + f$, and both u, v are on whiskers. (See Figure 4).

Case II : $a \leq b$ and $a \leq c \leq a + f$. Then $uy = a$, and u is on a whisker; and $vz = a + f$, $vy = -a + b + c$, and v is located along the side of the rectangle between y and z. (See Figure 5).

Case III : $b \leq a \leq b + e$ and $b \leq c \leq b + f$. Then $ux = b + e$, $uy = a$, and u is located along the side of the rectangle between x and y; and $vz = b + f$, $vy = c$, and v is located along the side of the rectangle between y and z (See Figure 6).

Case IV : $b \le a \le b+e$ and $b+f \le c$. Then $ux = b+e$, $uy = a$, and u is located on the side of the rectangle between x and y; and $vz = b+f$, and v is on a whisker. (See Figure 7).

Case V : $b+e \le a$ and $b+f \le c$. Then $ux = b+e$, $vz = b+f$, and both are on whiskers. (See Figure 8).

All other possibilities are similar to one of the above five cases.

The adversary pseudo-cost for each case is

$$\nabla(S, r, S') = \min_{X \ni r} \{SX - S'X\},$$

where X may be taken to be one of $\{r, x, y\}$, $\{r, x, z\}$, or $\{r, y, z\}$. For Cases I and II, each of the six quantities (three choices of X for each case) works out to be $d - a$, while for Cases III through V, each of the nine quantities works out to be $d - b$. Thus, the adversary pseudo-cost is always $d - \min\{a, b, c\}$.

Finally, the minimum matching of S with S' determines the motion of our servers. We show that the thesis of the lemma is true in each of the five cases:

$\diamondsuit$ If x is closest to r, then x moves to r, y moves to u, and z moves to v, and

$$xr - yu - zv = \begin{cases} d-a & \text{in Cases I, II} \\ d-b & \text{in Cases III, IV.} \end{cases}$$

$\diamondsuit$ If y is closest to r, then y moves to r, x moves to u, and z moves to v, and

$$yr - xu - zv = d - b \text{ in Case V.}$$

$\diamondsuit$ If z is closest to r, then z moves to r, x moves to u, and y moves to v, and

$$yr - xu - zv = d - a \text{ in Case II.}$$

All other situations are similar to one of the above. This completes the proof of Lemma 2.5. $\diamond$

Theorem 2.1 *Equipoise is 2-competitive for two servers.*

Proof: We define the potential to be $\Phi = xy$, where x, y are our server positions.

Now consider one move. Let a, b, c be the parameters of $\mathbf{CH}(K)$, as before, where $a \le b$. The adversary pseudo-cost of the move is c by Lemma 2.5, our cost is $2a + c$, and the change in potential is $c - 2a$. Thus, denoting by Φ and Φ' the value of the potential before and after the move we have

$$cost_{EQ}(S, r) + \Phi' = 2 \cdot \nabla(S, r, S') + \Phi,$$

and we are done, by Lemma 2.4. $\diamond$

Theorem 2.2 *Equipoise is 11-competitive for three servers.*

Proof: We will reduce the problem of proving that our cost is 11-competitive against the adversary pseudo-cost by reduction to a "Blue–Red game," a problem on a bi-weighted directed graph. We then show that this graph has certain "prime" edges, such that every edge "factors" as a path involving only prime edges. We then define a potential function on the vertices of the graph, and show that, for each prime edge, the "Blue" cost plus the increase of the potential does not exceed eleven times the "Red" cost.

The Blue–Red game. We define a *Blue–Red game* to be a directed graph, $\mathcal{G}$, with two costs, the *Blue cost* and the *Red cost*, associated with every edge. We say that $\mathcal{G}$ is *c-competitive* if there exists a function Φ, defined on the vertices of $\mathcal{G}$, such that, for every vertex ν and every path α which starts at ν

$$cost_{Blue}(\alpha) \leq c \cdot cost_{Red}(\alpha) + \Phi(\nu)$$

We let the vertices of $\mathcal{G}$ be all multisets consisting of three non-negative real numbers, not necessarily distinct. (The triple $\{\ell_1, \ell_2, \ell_3\}$ represents the slack parameters of the convex hull of our server positions.) There are five classes of edges of $\mathcal{G}$ (corresponding to the five cases of moves of Equipoise). Let a, b, c, d, e, f be any non-negative real numbers. Then there are edges of $\mathcal{G}$ from $\nu = \{a + e, b, c + f\}$ as follows:

$\diamondsuit$ Class I : If $a \leq b$ and $a + f \leq c$, there is an edge

$$\{a + e, b, c + f\} \to \{d + e, b - a + f, c - a - f\}$$

with Red cost $d - a$ and Blue cost $3a + d + 2f$ (see Fig. 4).

$\diamondsuit$ Class II : If $a \leq b$ and $a \leq c \leq a + f$, there is an edge

$$\{a + e, b, c + f\} \to \{0, b + c - 2a, a - c + d + e + f\}$$

with Red cost $d - a$ and Blue cost $\min\{3a + d + 2f, a + 2c + d + 2e\}$ (see Fig. 5).

$\diamondsuit$ Class III : If $b \leq a \leq b + e$ and $b \leq c \leq b + f$, there is an edge

$$\{a + e, b, c + f\} \to \{a - b, c - b, d - a - c + 2b + e + f\}$$

with Red cost $d - b$ and Blue cost $\min\{2a + b + d + 2f, b + 2c + d + 2e\}$ (see Fig. 6).

$\diamondsuit$ Class IV : If $b \leq a \leq b + e$ and $b + f \leq c$, there is an edge

$$\{a + e, b, c + f\} \to \{f, d - a + b + e, a + c - 2b - f\}$$

with Red cost $d - b$ and Blue cost $2a + b + d + 2f$ (see Fig. 7).

$\diamondsuit$ Class V : If $b+e \le a$ and $b+f \le c$, there is an edge

$$\{a+e, b, c+f\} \to \{d, a-b-e+f, c-b+e-f\}$$

with Red cost $d-b$ and Blue cost $3b+2d+2e+f$ (see Fig. 8).

Prime moves. We will identify a set of edges in $\mathcal{G}$, which we call *prime* edges, such that every edge can be *factored* into prime edges. Precisely, a *prime factorization* of an edge $\nu_1 \to \nu_2$ is a path from ν_1 to ν_2, consisting of entirely prime edges, whose total Red (Blue) cost is equal to the Red (Blue) cost of $\nu_1 \to \nu_2$.

We now list the prime moves, of which there are four types. Each prime move is a special case of one of the five classes of moves listed above.

(A) If $\varepsilon \le \ell_i$ for each i then

$$\{\ell_1, \ell_2, \ell_3\} \stackrel{A}{\to} \{\ell_1 - \varepsilon, \ell_2 - \varepsilon, \ell_3 - \varepsilon\}.$$

(B)

$$\{\ell_1, \ell_2, \ell_3\} \stackrel{B}{\to} \{\ell_1 + \varepsilon, \ell_2, \ell_3\}.$$

(C) If $2\varepsilon \le \ell_3$ then

$$\{\ell_1, \ell_2, \ell_3\} \stackrel{C}{\to} \{\ell_1 + \varepsilon, \ell_2, \ell_3 - 2\varepsilon\}.$$

(D)

$$\{\ell_1, \ell_2, \ell_3\} \stackrel{D}{\to} \{0, \ell_1 + \ell_2, \ell_3\}.$$

The five classes of moves factor into primes as follows.

(I) Move I factors into primes as

$$\begin{aligned} \{a+e, b, c+f\} &\stackrel{A}{\to} \{e, -a+b, -a+c+f\} \\ &\stackrel{C}{\to} \{e, -a+b+f, -a+c-f\} \\ &\stackrel{B}{\to} \{d+e, -a+b+f, -a+c-f\} \end{aligned}$$

(II) Move II factors into primes as

$$\begin{aligned} \{a+e, b, c+f\} &\stackrel{A}{\to} \{e, -a+b, -a+c+f\} \\ &\stackrel{C}{\to} \{e, -2a+b+c, a-c+f\} \\ &\stackrel{D}{\to} \{a-c+e+f, -2a+b+c, 0\} \\ &\stackrel{B}{\to} \{a-c+d+e+f, -2a+b+c, 0\} \end{aligned}$$

(III) Provided $a + f \leq c + e$, *i.e.*, the server at x moves to r, Move III factors into primes as

$$\begin{aligned}
\{a+e, b, c+f\} &\xrightarrow{A} \{a-b+e, 0, -b+c+f\} \\
&\xrightarrow{C} \{a-b+e, -b+c, b-c+f\} \\
&\xrightarrow{D} \{a-c+e+f, -b+c, 0\} \\
&\xrightarrow{C} \{-a+2b-c+e+f, -b+c, a-b\} \\
&\xrightarrow{B} \{-a+2b-c+d+e+f, -b+c, a-b\}
\end{aligned}$$

The case where $a + f \geq c + e$, *i.e.*, the server at z moves to r, is symmetric.

(IV) Move IV factors into primes as

$$\begin{aligned}
\{a+e, b, c+f\} &\xrightarrow{A} \{a-b+e, 0, -b+c+f\} \\
&\xrightarrow{C} \{a-b+e, f, -b+c-f\} \\
&\xrightarrow{C} \{-a+b+e, f, a-2b+c-f\} \\
&\xrightarrow{B} \{-a+b+d+e, f, a-2b+c-f\}
\end{aligned}$$

(V) Move V factors into primes as

$$\begin{aligned}
\{a+e, b, c+f\} &\xrightarrow{A} \{a-b+e, 0, -b+c+f\} \\
&\xrightarrow{C} \{a-b-e, 0, -b+c+e+f\} \\
&\xrightarrow{C} \{a-b-e+f, 0, -b+c+e-f\} \\
&\xrightarrow{B} \{a-b+d-e+f, 0, -b+c+e-f\}
\end{aligned}$$

It is a routine task to check that both the Blue and the Red costs add up correctly.

Lemma 2.6 *The Blue–Red game $\mathcal{G}$ is* 11*-competitive.*

Proof: Consider a vertex $\nu = \{\ell_1, \ell_2, \ell_3\}$ of $\mathcal{G}$, where the ℓ_i are ordered so that $\ell_1 \leq \ell_2 \leq \ell_3$. Then the potential of ν is defined as

$$\Phi(\nu) = 10 \cdot \ell_1 + 8 \cdot \ell_2 + 6 \cdot \ell_3.$$

We claim that, for any edge $\nu_1 \to \nu_2$ of $\mathcal{G}$,

$$cost_{Blue}(\nu_1 \to \nu_2) + \Phi(\nu_2) - \Phi(\nu_1) \;\leq\; 11 \cdot cost_{Red}(\nu_1 \to \nu_2). \tag{1}$$

Since every edge can be factored into primes, it suffices to prove (1) only for prime edges. We consider each of the four primes:

Move A: the Red cost is $-\varepsilon$, the Blue cost is 3ε, and the potential decreases by 24ε.

Move B: the Red cost is ε, the Blue cost is ε, and the potential increases by at most 10ε. (The worst case is that the shortest arm grows.)

Move C: the Red cost is 0, the Blue cost is 2ε, and the potential decreases by at least 2ε.

Move D: the Red cost is 0. Without loss of generality, $\ell_1 \le \ell_2$, so the Blue cost is $2\ell_1$, and the potential decreases by at least $2\ell_1$. (There are two worst cases: $\ell_1 + \ell_2 \le \ell_3$, and $\ell_3 = 0$.)

Thus, (1) is true for all edges.

Finally, if α is any path in $\mathcal{G}$ starting from a vertex ν, we obtain, by adding up (1) for all edges of α,

$$cost_{Blue}(\alpha) \le 11 \cdot cost_{Red}(\alpha) + \Phi(\nu)$$

which completes the proof of Lemma 2.6. $\diamond$

We now continue the proof of Theorem 2.2. If $S \in \Lambda^3\mathbf{CH}(M)$ has slack parameters ℓ_1, ℓ_2, ℓ_2, we associate S with $\nu = \{\ell_1, \ell_2, \ell_2\}$, a vertex of the Blue–Red game $\mathcal{G}$. Similarly, we associate S' with $\nu' = \{\ell_1', \ell_2', \ell_2'\}$. By the way the costs of the edges of $\mathcal{G}$ are defined, and by Lemma 2.5, $cost_{Blue}(\nu \to \nu')$ is equal to $cost_{EQ}(S, r) = SS'$, and $cost_{Red}(\nu \to \nu')$ is equal to $\nabla(S, r, S')$. By Lemmas 2.4 and 2.6, we are done. $\diamond$

2.2.1 Lower Bound

By actual example, we can show that Lemma 2.6 is tight. Our method is to define a cycle in $\mathcal{G}$ whose Blue cost is at least $(11 - \varepsilon)$ times its Red cost, for any real $\varepsilon > 0$. This does not prove that Equipoise is no better than 11-competitive for 3 servers, but since that algorithm was designed so as to minimize our cost relative to adversary pseudo-cost, it seems to indicate that a better competitiveness for Equipoise will not be easily found.

Lemma 2.7 *For any $\varepsilon > 0$ and any $\ell \ge 2\varepsilon$, there is a path in $\mathcal{G}$ from $\{0, 0, \ell\}$ to $\{\varepsilon, \varepsilon, \ell - 2\varepsilon\}$ whose Blue cost is 5ε and whose Red cost is ε.*

Proof: The path is

$$\begin{aligned} \{0, 0, \ell\} &\xrightarrow{C} \{0, \varepsilon, \ell - 2\varepsilon\} \\ &\xrightarrow{B} \{\varepsilon, \varepsilon, \ell - 2\varepsilon\} \\ &\xrightarrow{D} \{0, 2\varepsilon, \ell - 2\varepsilon\} \end{aligned}$$

$\diamond$

Lemma 2.8 *For any $\varepsilon > 0$ and any $\ell_1 \ge \varepsilon, \ell_2 \ge 2\varepsilon$, there is a path in $\mathcal{G}$ from $\{0, \ell_1, \ell_2\}$ to $\{0, \ell_1 + 2\varepsilon, \ell_2 - 2\varepsilon\}$ whose Blue cost is 7ε and whose Red cost is ε.*

Proof: The path is

$$\begin{aligned}\{0,\ell_1,\ell_2\} &\xrightarrow{C} \{\varepsilon,\ell_1,\ell_2-2\varepsilon\}\\ &\xrightarrow{D} \{0,\ell_1+\varepsilon,\ell_2-2\varepsilon\}\\ &\xrightarrow{B} \{\varepsilon,\ell_1+\varepsilon,\ell_2-2\varepsilon\}\\ &\xrightarrow{D} \{0,\ell_1+2\varepsilon,\ell_2-2\varepsilon\}\end{aligned}$$

$\diamond$

Theorem 2.3 *The Blue-Red graph $\mathcal{G}$ is no better than* 11*-competitive.*

Proof: Suppose $\mathcal{G}$ is c'-competitive for some $c' < c$. Let $\varepsilon = \frac{c'-c}{4}$. We construct a cycle, ζ, based at $\{0,0,1\}$, whose Red cost is $\frac{1}{4}$ and whose Blue cost is greater than $\frac{11}{4} - \varepsilon$. Let $n > \frac{1}{2\varepsilon}$ be an integer. Use Lemma 2.7 to construct a path α from $\{0,0,1\}$ to $\{0, \frac{1}{2n}, 1-\frac{1}{2n}\}$ whose Blue cost is $\frac{5}{4n}$ and whose Red cost is $\frac{1}{4n}$. Then use Lemma 2.8 to construct a path β^i from $\{0, \frac{i-1}{2n}, 1-\frac{i-1}{2n}\}$ to $\{0, \frac{i}{2n}, 1-\frac{i}{2n}\}$, for $i = 2,3,\ldots n$, whose Blue cost is $\frac{7}{4n}$ and whose Red cost is $\frac{1}{4n}$. Finally, let γ be the edge $\{0,\frac{1}{2},\frac{1}{2}\} \to \{0,0,1\}$, which has Blue cost 1 and Red cost 0. The cycle $\zeta = \alpha\beta^2\beta^3\ldots\beta^n\gamma$ has Blue cost $\frac{11}{4} - \frac{1}{2n}$ and Red cost $\frac{1}{4}$. Then

$$\lim_{m\to\infty} \left(cost_{Blue}(\zeta^m) - c' \cdot cost_{Red}(\zeta^m)\right) = \infty$$

(where ζ^m denotes the m-fold concatenation of ζ), contradicting c'-competitiveness. $\diamond$

2.3 The Work Function Algorithm for k Servers

In this subsection we introduce what we call the "work function" strategy for the k-server problem.

This strategy has been independently proposed by Howard Karloff, who called it the "Opt-based" algorithm, and by McGeoch and Sleator, who called it the "Better-Late-than-Never" algorithm.

We define a *work function* to be a non-negative real-valued function $\omega : \Lambda^k M \to \mathbf{R}^+$ (where $\mathbf{R}^+$ is the set of non-negative reals) such that, for any two server configurations X and Y

$$\omega(X) - \omega(Y) \leq XY.$$

We call this the *slope condition*. We can think of $\omega(X)$ as a conditional obligation of the adversary – the amount of cost we can assume the adversary must have incurred if he is at configuration X. By $\mathbf{W}_M$ we denote the set of all work functions (for simplicity we will write $\mathbf{W}$ when M is understood).

Support. If ω is a work function and X, Y are configurations, we say that X *dominates* Y if $\omega(Y) = \omega(X) + XY$, *i.e.*, the conditional obligation of the adversary if he is at Y is the

maximum possible consistent with his obligation if he is at X. When that holds, the adversary is always at least as well off to be at X as to be at Y. We say that X is a *support configuration* of ω if it is not dominated by any other configuration, and we say that ω is *finitary* if $Supp(\omega)$, the set of all support configurations, is finite, and if every configuration is dominated by some support configuration. If ω is finitary, it is characterized by its values on its support, and an adversary can do no better than to always have its servers at a support configuration.

We say that ω is a *cone* if $Supp(\omega)$ has just one element. If a work function is a cone, an algorithm "knows" the location of the adversary servers. A cone with support X will be denoted by χ_X or often simply by X if it does not lead to confusion. Clearly, $\chi_X(Y) = XY$ or, using the simplified notation, $X(Y) = XY$. Sometimes we also refer to χ_X as the *characteristic function* of X.

The update operator. If ω is a work function and $r \in M$, we define a work function $\omega \wedge r$, the *update of* ω *by* r, by

$$(\omega \wedge r)(X) = \min_{Y \ni r} \{\omega(Y) + XY\}.$$

An equivalent formulation of the update operator, which seems more complex, but is actually easier to use, is

$$(\omega \wedge r)(X) = \begin{cases} \omega(X) & \text{if } r \in X \\ \min_{Y \ni r} \{(\omega \wedge r)(Y) + XY\} & \text{otherwise} \end{cases}$$

Remark 2.1

(a) *In either formulation of* $\omega \wedge r$*, the minimum can be taken over only the* k *choices of* Y *which lie in* $X \cup \{r\}$*.*

(b) *If* $X \in Supp(\omega \wedge r)$*, then* $r \in X$*.*

(c) *If* ω *is finitary, then* $\omega \wedge r$ *is finitary.*

Informally, if ω is the system of conditional obligations of the adversary, then $\omega \wedge r$ is the new system of conditional obligations after one more request, r. The update operator can be extended to arbitrary sequences by setting $\omega \wedge \epsilon = \omega$ (where ϵ is the empty sequence) and $\omega \wedge (\varrho r) = (\omega \wedge \varrho) \wedge r$.

Most often it is convenient to deal with functions whose infimum is zero, and thus we also define another operator

$$(\omega \triangle r)(X) = (\omega \wedge r)(X) - \inf(\omega \wedge r).$$

As with $\wedge$, the operator $\triangle$ can be extended to sequences of requests in an obvious way. In order to simplify the notation we will often omit the parenthesis and write $\omega \wedge \varrho(X)$ and $\omega \triangle \varrho(X)$.

By an easy inductive argument on m, we have:

Remark 2.2 *If $X \in \Lambda^k M$, $\varrho = r^1 \dots r^m \in M^*$, and $Y \in Supp(X \wedge \varrho)$, then*

$$r \in Y \subseteq X \cup \{r^1, \dots, r^m\}.$$

We can now define the Work Function Algorithm (WFA).

The algorithm keeps track of a *current work function* ω. Initially the servers are on some configuration S^0 and $\omega = S^0$ (remember that we identify $X \in \Lambda^k M$ with its characteristic function χ_X).

The Work Function Algorithm: Let S be the current server configuration, and r the new request point. Then update the current work function by

$$\omega \leftarrow \omega \wedge r$$

and move one server to r so that the quantity

$$SS' + \omega(S')$$

is minimized, where S' is the configuration after the move.

Alternatively, ω can be the current offset function, since the difference is a constant. In this case, the update step is $\omega \leftarrow \omega \triangle r$.

Interestingly, the update step of WFA can be done after S' is chosen instead of before, or in parallel. The reason is that $\omega \wedge r(X) = \omega(X)$ for any $X \ni r$.

Time and space complexity. The space complexity of WFA is the maximum cardinality of $Supp(\omega)$, over all offset functions ω that occur during the computation. If $|M| = n$, that does not exceed the number of multisets in M of order k which contain a specific point, which is $\binom{n+k}{k-1}$, while if M is infinite, it does not exceed $\binom{m+k-1}{k-1}$ (m = the number of requests). The time complexity is dominated by the time needed to update the current work function. If $|M| = n$, that time complexity is $\tilde{O}(k\binom{n+k}{k-1})$ for each step. (The notation $\tilde{O}$ conceals a polylogarithmic factor.) If M is infinite, it is $\tilde{O}(k\binom{t+k-1}{k-1})$ for the t^{th} step.

Current Status of WFA. At the present time, we are able to prove that WFA is 2-competitive for $k = 2$. We have been unable to prove that WFA is k-competitive for higher k, but computer simulations using tens of thousands of small metric spaces have uncovered no counter-example.

In Section 2.3.1 we give what the outline of what we believe will eventually be the proof of k-competitiveness of the WFA for the k-server problem.[3]

[3] At the present time, all that remains to do is to prove one conjecture that seems "intuitively obvious."

2.3.1 A Conjectured Outline of the Proof of Optimality

Pseudo-cost. Let $\mathbf{W}^0 \subseteq \mathbf{W}$ be the set of all offset functions, *i.e.*, all work functions whose infimum is zero. If $\omega \in \mathbf{W}^0$ is any offset function and $r \in M$ is any request, we define the *pseudo-cost* of the request r on ω to be [4]

$$\nabla(\omega, r) = \sup_X \{\omega \triangle r(X) - \omega(X)\}.$$

Remark 2.3 *That supremum is always achieved for some $X \in Supp(\omega)$, if ω is finitary.*

Proof: $\omega \wedge r(X) - \omega(X) \geq \omega \wedge r(Y) - \omega(Y)$ if X dominates Y. $\diamond$

Intuitively, the pseudo-cost is the "worst-case" increase in the conditional obligation for a given request, minus the optimal cost.

If $\varrho = r^1 \dots r^m$ is any request sequence, define

$$\nabla(\omega, \varrho) = \sum_{t=1}^{m} \nabla(\omega \triangle r^1 \dots r^{t-1}, r^t)$$

Lemma 2.9 *If $X \in \Lambda^k M$ and $\varrho \in M^*$ then $cost_{WFA}(X, \varrho) \leq \nabla(X, \varrho)$.*

Proof: Let $\varrho = r^1 r^2 \dots r^m$, $S^t = WFA(S^0, r^1 \dots r^t)$ and $\omega^t = S^0 \triangle r^1 \dots r^t$.

Claim A: $\omega^t(S^{t-1}) = \omega^t(S^t) + S^{t-1}S^t$ for each t.

Since $\omega^t = \omega^{t-1} \triangle r^t$, we have $\omega^t(S^{t-1}) = \min_{X \ni r^t}\{\omega^t(X) + S^{t-1}X\}$. That minimum is achieved for $X = S^t$ by definition of WFA, proving Claim A. Now $\omega^0(S^0) = 0$ by definition. Thus

$$\begin{aligned} cost_{WFA}(\varrho) &= \sum_{t=1}^{m} S^{t-1}S^t \leq \sum_{t=1}^{m} \left[S^{t-1}S^t + \omega^t(S^t) - \omega^{t-1}(S^{t-1})\right] \\ &= \sum_{t=1}^{m} \left[\omega^t(S^{t-1}) - \omega^{t-1}(S^{t-1}]\right) \leq \sum_{t=1}^{m} \nabla(\omega^{t-1}, r^t) = \nabla(S^0, \varrho) \end{aligned}$$

$\diamond$

Lemma 2.10 *Suppose that there is a function $\Phi : \mathbf{W}^0 \to \mathbf{R}^+$ such that, for any offset function ω and any request r, $\nabla(\omega, r) + \Phi(\omega \triangle r) \leq \Phi(\omega) + k \cdot \inf(\omega \wedge r)$. Then WFA is k-competitive.*

Proof: The result follows easily from Lemma 2.9 and summation over the request sequence. Let $\varrho = r^1 \dots r^m$ and $\omega^t = S^0 \triangle r^1 \dots r^{t-1}$. By definition of the operators $\wedge$ and $\triangle$, we have

$$cost_{opt}(S^0, \varrho) = \inf(S^0 \wedge \varrho) = \sum_{t=1}^{m} \inf(\omega^{t-1} \wedge r^t).$$

[4] This is *our* pseudo-cost, an estimate of our cost, not to be confused with the *adversary pseudo-cost* defined in 2.2.

Then, by Lemma 2.9,

$$\begin{aligned} cost_{WFA}(X, \varrho) &\leq \nabla(X, \varrho) = \sum_{t=1}^{m} \nabla(\omega^{t-1}, r^t) \\ &\leq \sum_{t=1}^{m} \left[\Phi(\omega^{t-1}) - \Phi(\omega^t) + k \cdot \inf(\omega^{t-1} \wedge r^t)\right] \\ &\leq k \cdot cost_{opt}(S^0, \varrho) + \Phi(\omega^0) \end{aligned}$$

The k-competitiveness of WFA follows from Lemma 2.9. $\diamond$

A potential that is conjectured to work. For any configuration $X = \{x_1, x_2 \ldots x_k\}$, define ΣX to be the total edge length of the complete graph on X, namely $\Sigma X = \sum_{1 \leq i < j \leq k} x_i x_j$. If ω is an offset function and X is a configuration, define

$$\Phi_X(\omega) = \Sigma X + \sup_{\rho \in X^*} \nabla(\omega, \rho) - k \cdot \omega(X).$$

(In this context, X^* is the set of all strings whose symbols are members of X, *i.e.*, request sequences consisting entirely of requests drawn from the k members of X.) Then, for any offset function ω, define

$$\Phi(\omega) = \sup_X \Phi_X(\omega).$$

Conjecture 2.1 *The function Φ defined above satisfies the hypotheses of Lemma 2.10.*

Conjecture 2.2 *Let $\omega = S^0 \triangle \varrho r$ be an offset function obtained via a sequence of requests of which the last one is r. Then there exists a configuration X such that $r \in X$ and $\Phi(\omega) = \Phi_X(\omega)$.*

Lemma 2.11 *Conjecture 2.2 implies Conjecture 2.1.*

Proof: $\Phi(\omega, r)$ is non-negative because $\inf \omega = 0$. Let $\epsilon > 0$. Pick a configuration $X \ni r^t$ such that

$$\Phi(\omega^t) < \Phi_X(\omega^t) + \epsilon$$

and pick $\pi \in X^\star$ such that

$$\Phi_X(\omega^t) < \Sigma X + \nabla(\omega^t, \pi) - k \cdot \omega^t(X) + \epsilon.$$

Then

$$\begin{aligned} \Phi(\omega^t) + \nabla(\omega^{t-1}, r^t) &< \Phi_X(\omega^t) + \nabla(\omega^{t-1}, r^t) + \epsilon \\ &< \Sigma X + \nabla(\omega^{t-1}, r^t) + \nabla(\omega^t, \pi) - k \cdot \omega^t(X) + 2\epsilon \\ &= \Sigma X + \nabla(\omega^{t-1}, r^t \pi) - k \cdot \omega^{t-1}(X) + k \cdot \inf(\omega^{t-1} \wedge r^t) + 2\epsilon \\ &\leq \Phi_X(\omega^{t-1}) + k \cdot \inf(\omega^{t-1} \wedge r^t) + 2\epsilon \\ &\leq \Phi(\omega^{t-1}) + k \cdot \inf(\omega^{t-1} \wedge r^t) + 2\epsilon \end{aligned}$$

Letting $\epsilon \to 0$, we obtain the hypothesis of Lemma 2.10. $\diamond$

Conjecture 2.3 *Let $\omega = S^0 \triangle \varrho$, for some configuration $S^0 \in \Lambda^k M$ and string of requests $\varrho \in M^*$. Then there exists $X \in Supp(\omega)$ such that $\Phi(\omega) = \Phi_X(\omega)$*

Conjecture 2.3 implies Conjecture 2.2, since $r \in X$ for any $X \in Supp(\omega)$, and hence implies that WFA is k-competitive.

Computer Testing. We have tested Conjecture 2.3 on a computer. No counter-example has been found, searching tens of thousands of small metric spaces.

2.3.2 WFA is 2-Competitive for 2 Servers

Theorem 2.4 *WFA is 2-competitive for $k = 2$.*

Proof: We do not prove Conjecture 2.1. Instead, we construct a different potential which satisfies the hypothesis of Lemma 2.10.

Let r be the last request and ω the current offset function. For simplicity we will write $\omega(x, y)$ instead of $\omega(\{x, y\})$. The potential function is

$$\Phi = \Phi(\omega, r) = \max_{x,p,q} \{\, rx - \omega(r, x) + rp - \omega(x, p) + xq - \omega(r, q) \,\}.$$

We first need to show that $\Phi(\omega, r) \geq 0$. Since $\inf \omega = 0$ and ω is finitary, $\omega(r, a) = 0$ for some a. Letting $x = p = q = a$ in the definition shows that $\Phi(\omega, r) \geq 0$.

We claim that Φ satisfies the hypothesis of Lemma 2.10, for $k = 2$.

In the proof we will use without explicit reference the equality

$$\omega(x, y) = \min\{\omega(r, x) + ry, \omega(r, y) + rx\}$$

that follows from Remark 2.1 (a). We first derive a somewhat different formula for Φ:

$$\begin{aligned}
\Phi &= \max_{x,p,q} \left\{ rx - \omega(r, x) + rp - \min \left\{ \begin{array}{c} \omega(r, x) + rp \\ \omega(r, p) + rx \end{array} \right\} + xq - \omega(r, q) \right\} \\
&= \max_{x,p,q} \max \left\{ \begin{array}{c} xq - \omega(r, q) - 2 \cdot \omega(r, x) + rx \\ xq - \omega(r, q) - \omega(r, x) + rp - \omega(r, p) \end{array} \right\} \\
&= \max_{x,p,q} \{xq - \omega(r, q) - \omega(r, x) + rp - \omega(r, p)\}. \qquad (2)
\end{aligned}$$

The last equality follows from the fact that the expression in the top line inside the maximum is a special case of the one on the bottom line when $p = x$.

The advantage of formula (2) is that it is symmetric under the exchange of x and q, reducing the number of cases to consider in our proof.

Let r be the previous request point and r' the new request point. Let ω denote the offset function after request r, and let $\omega' = \omega \wedge r'$. Remark 2.1(a) implies that $\omega'(r', t) = \omega(r', t)$ for

each $t \in M$. This fact will be used several times in the proof. Denoting by Φ' the potential after the request, by formula (2) we have

$$\begin{aligned} \Phi' &= \max_{y,u,v}\{yv - \omega'(r',v) - \omega'(r',y) + r'u - \omega'(r',u)\} + 3 \cdot \inf \omega' \\ &= \max_{y,u,v}\{yv - \omega(r',v) - \omega(r',y) + r'u - \omega(r',u)\} + 3 \cdot \inf \omega'. \end{aligned} \tag{3}$$

We also have

$$\omega'(r,z) = \min \left\{ \begin{array}{c} \omega'(r',z) + rr' \\ \omega'(r',r) + r'z \end{array} \right\} = \min \left\{ \begin{array}{c} \omega(r',z) + rr' \\ \omega(r',r) + r'z \end{array} \right\} = \min \left\{ \begin{array}{c} \omega(r,z) + 2 \cdot rr' \\ \omega(r,r') + r'z \end{array} \right\},$$

and therefore, using also Remark 2.3, we obtain $\nabla(\omega, r') + \inf \omega' = \max G(z)$, where

$$G(z) = \omega'(r,z) - \omega(r,z) = \min\{2 \cdot rr', r'z + \omega(r,r') - \omega(r,z)\}.$$

Let $y, u, v, z \in M$ be arbitrary. We now claim that

$$yv - \omega(r',v) - \omega(r',y) + r'u - \omega(r',u) + G(z) \leq \Phi. \tag{4}$$

Let LS denote the left-hand side of (4). By symmetry, we have three cases:

Case 1: $\omega(r',v) = \omega(r,v) + rr'$ and $\omega(r',y) = \omega(r,y) + rr'$. We note that, for any $a, b \in M$, $ab - \omega(a,b) \leq rc - \omega(r,c)$, where $c = a$ or $c = b$. Then

$$\begin{aligned} \text{LS} &\leq yv - \omega(r,v) - rr' - \omega(r,y) - rr' + r'u - \omega(r',u) + 2 \cdot rr' \\ &\leq yv - \omega(r,v) - \omega(r,y) + rc - \omega(r,c) \ \leq \ \Phi. \end{aligned}$$

where c is either r' or u.

Case 2: $\omega(r',v) = \omega(r,r') + rv$ and $\omega(r',u) = \omega(r,u) + rr'$. Then

$$\begin{aligned} \text{LS} &\leq yv - \omega(r,r') - rv - \omega(r',y) + r'u - \omega(r,u) - rr' + 2 \cdot rr' \\ &\leq ry - \omega(r,r') + rr' + r'u - \omega(r',y) - \omega(r,u) \ \leq \ \Phi. \end{aligned}$$

Case 3: $\omega(r',v) = \omega(r,r') + rv$ and $\omega(r',u) = \omega(r,r') + ru$. Then

$$\begin{aligned} \text{LS} &\leq yv - \omega(r,r') - rv - \omega(r',y) + r'u - \omega(r,r') - ru + \omega(r,r') + r'z - \omega(r,z) \\ &\leq ry + rr' + r'z - \omega(r,r') - \omega(r',y) - \omega(r,z) \ \leq \ \Phi. \end{aligned}$$

Thus, (4) is verified. From (3) and (4), and the fact that $\nabla(\omega, r') = \max_z G(z) - \inf \omega'$, we obtain

$$\begin{aligned} \nabla(\omega,r') + \Phi' &= \max_z G(z) + \max_{y,u,v}\{yv - \omega(r',v) - \omega(r',y) + r'u - \omega(r',u)\} + 2 \cdot \inf \omega' \\ &\leq \Phi + 2 \cdot \inf \omega', \end{aligned}$$

which is that Φ satisfies the hypothesis of Lemma 2.10 for $k = 2$. $\diamond$

3 Theory of On-Line Games

In this section we present a general model of on-line problems that we call *on-line games*. We believe that on-line games capture most of the on-line problems in which competitive analysis is applicable. Our goal is to express and explain in this model certain phenomena that have been observed in on-line problems, and to develop general techniques for constructing competitive algorithms. Similar attempts to describe on-line problems in terms of games have been already presented in [1, 2, 17], but besides some scattered results no general theory of such games has been yet developed.

First we consider games in which negative cost moves are allowed, and a special kind of on-line algorithms that are memoryless, *i.e.*, their behavior does not depend on the past, only on the current state. We concentrate on *finite-cost* algorithms whose total cost is never bigger than a certain constant dependent only on the initial state. For such algorithms we prove a number of results. One major results is the proof that the existence of a finite-cost algorithm is equivalent to the existence of a potential function. This sheds a new light on why the potential technique used in the competitiveness proofs is so effective, since it basically shows that it is, in more or less explicit form, unavoidable. Later we show that this potential function can be determined from the description of the game. In finite cases, this potential (and therefore also the algorithm) can be actually *computed* using our method. Almost all our computer experiments were based on this technique.

Then we consider games with non-negative costs, in which we are seeking c-competitive algorithms. The major result is that this problem can be reduced to the previous one. That is, given such a game $\mathcal{G}$ we can construct another game $\mathcal{G}'$ (with possibly negative costs), find a memoryless finite-cost strategy $\mathcal{A}'$ in $\mathcal{G}'$, and then transform it into a competitive strategy for $\mathcal{G}$. A result with a similar flavor was presented by Manasse *et al.* [15] in the context of the server problem. Those results are only preliminary, and work is still in progress.

An *on-line* game is a triple $\mathcal{G} = \langle Q, \mathcal{R}, f \rangle$, where

- Q is called a set of *states*.
- $\mathcal{R}$ is a set of *requests*.
- $f : Q \times \mathcal{R} \times Q \to \mathbf{R}$ is a cost function. ($\mathbf{R}$ = reals)

For the purpose of this draft, we will assume that the sets $Q, \mathcal{R}$ are finite. In the final version we will formulate less restrictive assumptions about $\mathcal{G}$ which will allow infinite sets. Roughly speaking, Q, $\mathcal{R}$ and f must satisfy certain topological properties that will ensure that some minima and maxima that we have in the proofs are actually achieved.

Intuitively, we view $\mathcal{G}$ as the following game. At each time step the adversary gives us a request $r \in \mathcal{R}$. If we are now in state $x \in Q$, we move to any state $y \in Q$ at cost $f(x, r, y)$. In

the next section we will consider strategies that never pay more than a certain constant $\phi(x^0)$, where x^0 is the initial state. In general, we are interested in strategies that guarantee that our cost never exceeds c times the optimal cost plus $\phi(x^0)$, a constant depending only on the initial configuration.

3.1 Memoryless Strategies

A *memoryless on-line strategy* for $\mathcal{G}$ is a function $\mathcal{A} : Q \times \mathcal{R} \to Q$. If the current state is $x \in Q$ and the request is $r \in \mathcal{R}$, the strategy instructs us to move to state $\mathcal{A}(x, r)$.

Such a strategy $\mathcal{A}$ can be extended to a function $\mathcal{A} : Q \times \mathcal{R}^* \to Q$ by setting

$$\begin{aligned} \mathcal{A}^*(x, \epsilon) &= x, \quad \text{and} \\ \mathcal{A}(x, \varrho r) &= \mathcal{A}(\mathcal{A}(x, \varrho), r). \end{aligned}$$

If the current state is x, we will be in state $\mathcal{A}(x, \varrho)$ after serving the request sequence ϱ. Since we are interested only in on-line strategies, we will simply refer to $\mathcal{A}$ as a memoryless strategy. Sometimes we will also use the term "algorithm" instead of "strategy".

With $\mathcal{A}$ we associate a mapping $\mathcal{A}^* : Q \times \mathcal{R}^* \to Q^*$ defined as follows:

$$\begin{aligned} \mathcal{A}^*(x, \epsilon) &= x, \qquad \text{and} \\ \mathcal{A}^*(x, \varrho r) &= \mathcal{A}^*(x, \varrho)\mathcal{A}(x, \varrho r). \end{aligned}$$

If ϱ has length m, $\mathcal{A}^*(x, \varrho)$ is the sequence, of length $m + 1$, of the states of the algorithm as it serves ϱ, starting from x.

For $\mathcal{A}$ we also define a *cost function*, $cost_{\mathcal{A}} : Q \times \mathcal{R}^* \to \mathbf{R}$ as follows. Let $\mathcal{A}^*(x^0, \varrho) = x^0 x^1 x^2 \ldots x^m$. Then

$$cost_{\mathcal{A}}(x^0, \varrho) = \sum_{t=1}^{m} f(x^{t-1}, r^t, x^t).$$

3.1.1 Finite-Cost Strategies and Potential

A memoryless strategy $\mathcal{A}$ is called *finite-cost* if

$$\forall x \in Q \ : \ \sup_{\varrho \in \mathcal{R}^*} \{cost_{\mathcal{A}}(x, \varrho)\} < \infty.$$

Let $\mathcal{P}^+$ be the set of functions $Q \to \mathbf{R}^+$.

Theorem 3.1 *A strategy $\mathcal{A}$ is finite-cost iff there is a function $\phi \in \mathcal{P}^+$ such that*

$$\forall x \in Q \ : \ \sup_{r \in \mathcal{R}} \{ f(x, r, \mathcal{A}(x, r)) + \phi(\mathcal{A}(x, r)) - \phi(x) \} \ \leq \ 0.$$

Proof: ($\Leftarrow$) Let $x^0 \ldots x^m = \mathcal{A}^*(x^0, \varrho)$. Then

$$\begin{aligned} cost_{\mathcal{A}}(x^0, \varrho) &= \sum_{t=1}^{m} f(x^{t-1}, r^t, x^t) \\ &= \sum_{t=1}^{m} \Big[(f(x^{t-1}, r^t, x^t) + \phi(x^t) - \phi(x^{t-1}) \Big] + \phi(x^0) - \phi(x^m) \\ &\leq \phi(x^0), \end{aligned}$$

implying immediately that $\mathcal{A}$ is finite-cost.

($\Rightarrow$) Assume $\mathcal{A}$ is finite-cost. Let $\phi(x) = \sup_{\varrho \in \mathcal{R}^*} cost_{\mathcal{A}}(x, \varrho)$, which is well defined because $\mathcal{A}$ is finite-cost, and non-negative because we can take ϱ to be the empty string of requests. Then, for each $r \in \mathcal{R}$, using request string $r\varrho$ in the definition of ϕ we have

$$\begin{aligned} \phi(x) &\geq \sup_{\varrho \in \mathcal{R}^*} cost_{\mathcal{A}}(x, r\varrho) \\ &= f(x, r, \mathcal{A}(x, r)) + \sup_{\varrho \in \mathcal{R}^*} cost_{\mathcal{A}}(\mathcal{A}(x, r), \varrho) \\ &= f(x, r, \mathcal{A}(x, r)) + \phi(\mathcal{A}(x, r)), \end{aligned}$$

and therefore ϕ satisfies the needed condition. $\diamond$

A function ϕ satisfying the condition given in Theorem 3.1 will be referred to as a *potential function for* $\mathcal{A}$. The next theorem shows how we can associate a potential function with a game $\mathcal{G}$, as opposed to a potential of a particular strategy, and how we can extract a finite-cost strategy from this potential function.

Theorem 3.2 *There exists a finite-cost memoryless strategy for $\mathcal{G}$ iff there exists a function $\phi \in \mathcal{P}^+$ such that*

$$\forall x \in Q \ : \ \sup_{r \in \mathcal{R}} \inf_{y \in Q} \{ f(x, r, y) + \phi(y) - \phi(x) \} \ \leq \ 0$$

Proof: ($\Rightarrow$) Suppose that $\mathcal{A}$ is competitive. Let ϕ be the potential obtained by Theorem 3.1, and substitute $y = \mathcal{A}(x, r)$ in the formula above.

($\Leftarrow$) If such ϕ exists, define $\mathcal{A}(x, r)$ as that $y \in Q$ for which $f(x, r, y) + \phi(y) - \phi(x) \leq 0$. (Recall that Q is finite.) $\diamond$

A function ϕ satisfying the condition given in Theorem 3.2 will be called a *potential function for game* $\mathcal{G}$.

Theorem 3.3 *If $\mathcal{G}$ has a finite-cost strategy (and thus a potential), then it has a unique minimal potential function.*

Proof: Let ϕ_0 be a function defined by $\phi_0(x) = \inf_{\phi} \phi(x)$, where the infimum is taken over all potential functions for $\mathcal{G}$. We need only show that ϕ_0 is also a potential function for $\mathcal{G}$.

Fix $x \in Q$. Let ϕ be the potential function for $\mathcal{G}$ such that $\phi(x) = \phi_0(x)$. There must exist a $y \in Q$ such that $f(x, r, y) + \phi(y) \leq \phi(x)$. Thus

$$f(x, r, y) + \phi_0(y) \;\leq\; f(x, r, y) + \phi(y) \;\leq\; \phi(x) \;=\; \phi_0(x).$$

$\diamond$

3.1.2 Fixed Point Approach

In this section we show a relationship between the theory of on-line games and the fixed point theory for functional spaces. As it turns out, a memoryless competitive strategy (or, more precisely, its potential), if it exists, is a fixed point of a certain mapping $\widehat{U}$. This gives an algorithm that, given a game $\mathcal{G}$, computes a finite-cost strategy for $\mathcal{G}$ if it exists. We apply this method routinely in our computer programs that we use in our research on on-line algorithms. In particular, this method enabled us to test the k-server conjecture for $k = 3, 4$ on a large number of finite metric spaces with integer distances.

First we show how, given a strategy $\mathcal{A}$, we can determine its potential function.

Given a strategy $\mathcal{A}$, define the *$\mathcal{A}$-update function* $\widehat{U}_{\mathcal{A}} : \mathcal{P}^+ \to \mathcal{P}^+$ as follows:

$$\begin{aligned} U_{\mathcal{A}}(\phi)(x) &= \sup_{r \in \mathcal{R}} \{f(x, r, \mathcal{A}(x, r)) + \phi(\mathcal{A}(x, r))\}, \text{ and} \\ \widehat{U}_{\mathcal{A}}(\phi)(x) &= \max\{\phi(x), U_{\mathcal{A}}(\phi)(x)\}. \end{aligned}$$

Theorem 3.4 *A memoryless strategy $\mathcal{A}$ is finite-cost iff $\widehat{U}_{\mathcal{A}}$ has a fixed point. Furthermore, that fixed point is a potential function for $\mathcal{A}$.*

Proof: ($\Leftarrow$) Suppose that ϕ is the fixed point of $\widehat{U}_{\mathcal{A}}$, *i.e.*, $\widehat{U}_{\mathcal{A}}(\phi) = \phi$. Then, for each $x \in Q$ and $r \in \mathcal{R}$, we have

$$\phi(x) = \widehat{U}_{\mathcal{A}}(\phi)(x) \geq U_{\mathcal{A}}(\phi)(x) \geq f(x, r, \mathcal{A}(x, r)) + \phi(\mathcal{A}(x, r)),$$

and by Theorem 3.1 we have that $\mathcal{A}$ is finite-cost.

($\Rightarrow$) If $\mathcal{A}$ is finite-cost, let ϕ be the potential for $\mathcal{A}$. Then for each $x \in Q$ we have

$$U_{\mathcal{A}}(\phi)(x) = \sup_{r \in \mathcal{R}} \{f(x, r, \mathcal{A}(x, r)) + \phi(\mathcal{A}(x, r))\} \;\leq\; \phi(x),$$

and therefore $\widehat{U}_{\mathcal{A}}(\phi) = \max\{U_{\mathcal{A}}(\phi), \phi\} = \phi$. $\diamond$

Let $\widehat{U}_{\mathcal{G}} : \mathcal{P}^+ \to \mathcal{P}^+$ be the *update function for $\mathcal{G}$*, defined as follows

$$\begin{aligned} U_{\mathcal{G}}(\phi)(x) &= \sup_{r \in \mathcal{R}} \inf_{y \in Q} \{f(x, r, y) + \phi(y)\}, \text{ and} \\ \widehat{U}_{\mathcal{G}}(\phi)(x) &= \max\{\phi(x), U_{\mathcal{G}}(\phi)(x)\}. \end{aligned}$$

Theorem 3.5 *Game $\mathcal{G}$ has a memoryless finite-cost strategy iff $\widehat{U}_{\mathcal{G}}$ has a fixed point.*

Proof: ($\Rightarrow$) Suppose that $\mathcal{A}$ is finite-cost and let ϕ be its potential function. Then for each $x \in Q$ we have

$$\begin{aligned} U_{\mathcal{G}}(\phi)(x) &= \sup_{r\in\mathcal{R}} \inf_{y\in Q} \{f(x,r,y) + \phi(y)\} \\ &\leq \sup_{r\in\mathcal{R}} \{f(x,r,\mathcal{A}(x,r)) + \phi(\mathcal{A}(x,r))\} \;\leq\; \phi(x), \end{aligned}$$

and therefore ϕ is a fixed point of $\widehat{U}_{\mathcal{G}}$.

($\Leftarrow$) Suppose now that $\widehat{U}_{\mathcal{G}}$ has a fixed point ϕ. Then we have that for each $x \in Q$

$$\sup_{r\in\mathcal{R}} \inf_{y\in Q} \{f(x,r,y) + \phi(y)\} \;\leq\; \phi(x),$$

which, by Theorem 3.2, implies the existence of a finite-cost memoryless strategy for $\mathcal{G}$, since Q is finite. $\diamond$

3.2 Competitive Strategies with Memory

Now we will generalize a notion of an on-line strategy so that it is allowed to use the information from the past in making the decision in the current move. We consider only *positive games*, *i.e.*, games $\mathcal{G} = \langle Q, \mathcal{R}, f\rangle$ where $f \geq 0$.

An *on-line strategy* for $\mathcal{G}$ is a function $\mathcal{A} : Q \times \mathcal{R}^* \to Q$. If x^0 is the initial state, and if ϱ is a request sequence, then $\mathcal{A}(x^0, \varrho)$ is the state that we will be in after serving the requests ϱ, if we follow the strategy $\mathcal{A}$. With such an on-line strategy we can associate a mapping $\mathcal{A}^* : Q \times \mathcal{R}^* \to Q^*$, defined in the same way as for memoryless strategies. If x^0 is the initial state, and if ϱ is a request sequence of length m, then $\mathcal{A}^*(x^0, \varrho)$ is the string consisting of all states up to the m^{th} step. Analogously, we also define the cost function $cost_{\mathcal{A}}$.

For each state x^0 and request sequence $\varrho = r^1 r^2 \ldots r^m$ we define the optimal cost of servicing ϱ starting from x^0, as follows:

$$cost_{opt}(x^0, \varrho) = \inf_{x^1 \ldots x^m \in Q^*} \sum_{t=1}^{m} f(x^{t-1}, r^t, x^t).$$

If $\mathcal{A}$ is an on-line strategy for a positive game $\mathcal{G}$, we say that $\mathcal{A}$ is *c-competitive*, for some constant $c \in \mathbf{R}^+$, if

$$\forall x \in Q \;:\; \sup_{\varrho\in\mathcal{R}^*} \{cost_{\mathcal{A}}(x,\varrho) - c \cdot cost_{opt}(x,\varrho)\} < \infty$$

3.2.1 Work Functions and Offsets

Let $\mathcal{G} = \langle Q, \mathcal{R}, f\rangle$ be a positive game. A function $\omega : Q \to \mathbf{R}$ is called a *work function*. Denote by $\mathbf{W}_{\mathcal{G}}$ the set of all work functions on $\mathcal{G}$. An *offset function* on $\mathcal{G}$ is a work function whose

infimum is zero. We denote the set of offset functions on $\mathcal{G}$ by $\mathbf{W}^0_{\mathcal{G}}$. (We will omit the subscript $\mathcal{G}$ when it does not lead to confusion).

For each $x \in Q$ the *characteristic function* of x is defined by

$$\chi_x(y) = \sup_{\varrho \in \mathcal{R}^*} \{\, cost_{opt}(x, \varrho) - cost_{opt}(y, \varrho) \,\}.$$

Notice that χ_x is well defined because for each $y \in Q$ we have

$$\chi_x(y) \;\leq\; \sup_{r \in \mathcal{R}\; z \in Q} \{\, f(x, r, z) - f(y, r, z) \,\} \;<\; \infty.$$

Furthermore, $\chi_x(y) \geq 0$, since we can let ϱ be the empty string, and $\chi_x(x) = 0$. Intuitively, if the current offset function is χ_x, then x is the optimal state in the sense that, independent of the future requests, being now in state x cannot be worse than in any other state. Q embeds in $\mathbf{W}^0$ by regarding each $x \in Q$ as a function χ_x.

If $\omega : Q \to \mathbf{R}$ and $r \in \mathcal{R}$, define new work functions $\overline{\omega}$, $\omega \wedge R$, $\omega \triangle R \;:\; Q \to \mathbf{R}$ by

$$\begin{aligned} \overline{\omega}(x) &= \omega(x) - \inf_{y \in Q} \omega(y) \\ (\omega \wedge r)(x) &= \inf_{y \in Q} \{\omega(y) + f(y, r, x)\} \\ \omega \triangle r &= \overline{\omega \wedge r} \end{aligned}$$

We can also define $\omega \wedge \varrho$ and $\omega \triangle \varrho$, for any $\varrho \in \mathcal{R}^*$, to be the iteration of the respective operator for the individual requests.

Fact 3.1 *Let $x \in Q$, $\rho \in \mathcal{R}^*$. Then*

$$cost_{opt}(x, \varrho) \;=\; \inf(\chi_x \wedge \varrho) \;=\; \sum_{t=0}^{m} \inf(\omega^{t-1} \wedge r^t)$$

where $\varrho = r^1 r^2 \ldots r^m$ and $\omega^t = \chi_x \triangle r^1 \ldots r^{t-1}$ for each t.

Define a function $f^0 : \mathbf{W}^0 \times \mathcal{R}^* \times \mathbf{W}^0 \to \mathcal{R}^+$ by

$$f^0(\omega, \varrho, \mu) = \sup(\omega \wedge \varrho - \mu).$$

The motivation is that we need a function f^0 that would describe an optimal "cost" of going from function ω to μ on request sequence ϱ. If $\mu = \omega \triangle \varrho$, then there is no problem, we just set $f^0(\omega, \varrho, \mu) = \inf(\omega \wedge \varrho)$. However, for the purpose of the proof of Theorem 3.6, f^0 must be defined for an arbitrary μ. The above idea is generalized by realizing that, for $\mu = \omega \triangle \varrho$, the cost $f^0(\omega, \varrho, \mu)$, as defined above, is equal to the value by which we need to decrease $\omega \wedge \varrho$ in order to make it less than or equal to μ on every state.

Fact 3.2 *Function f^0 has the following properties:*

$$\begin{aligned} f^0(\omega, \varrho, \omega \triangle \varrho) &= \inf(\omega \wedge \varrho) \\ f^0(\chi_x, \varrho, \chi_x \triangle \varrho) &= cost_{opt}(x, \varrho) \\ f^0(\omega, \varrho, \mu) &\geq f^0(\omega, \varrho, \omega \triangle \varrho). \end{aligned}$$

If c is a constant, we define a new on-line game $\mathcal{G}^c$

$$\mathcal{G}^c = (Q \times \mathbf{W}^0_{\mathcal{G}}, \mathcal{R}, f^c),$$

where $f^c((x,\omega), r, (y,\mu)) = f(x,r,y) - c \cdot f^0(\omega, r, \mu)$. We will refer to $\mathcal{G}^c$ as the *c-residual game* (the terminology consistent with [15, 14]), and to f^c as the *c-residue function*.

Lemma 3.1 *If $\omega, \mu \in \mathbf{W}$ and $\varrho \in \mathcal{R}^*$, then* $\sup(\omega \wedge \varrho - \mu \wedge \varrho) \leq \sup(\omega - \mu)$.

Proof: It is sufficient to prove the lemma for $\varrho = r \in \mathcal{R}$. In this case, for each $x \in Q$ we have

$$\begin{aligned} \omega\wedge r(x) &= \inf_{y\in Q}\{\omega(y) + f(y,r,x)\} \leq \inf_{y\in Q}\{\mu(y) + \sup(\omega-\mu) + f(y,r,x)\} \\ &= \inf_{y\in Q}\{\mu(y) + f(y,r,x)\} + \sup(\omega-\mu) = \mu\wedge r(x) + \sup(\omega-\mu) \end{aligned}$$

$\diamond$

Lemma 3.2 *For each $\omega, \mu \in \mathbf{W}^0$, $\varrho \in \mathcal{R}^*$ and $r \in \mathcal{R}$ we have*

$$f^0(\omega, \varrho r, \mu \triangle r) \leq f^0(\omega, \varrho, \mu) + \inf(\mu \wedge r).$$

Proof: It is sufficient to prove the lemma for $\varrho = \epsilon$ (the empty sequence of requests), in which case, applying Lemma 3.1, we have

$$\begin{aligned} f^0(\omega, r, \mu\triangle r) &= \sup(\omega\wedge r - \mu\triangle r) = \sup(\omega\wedge r - \mu\wedge r + \inf(\mu\wedge r)) \\ &= \sup(\omega\wedge r - \mu\wedge r) + \inf(\mu\wedge r) \\ &\leq \sup(\omega-\mu) + \inf(\mu\wedge r) = f^0(\omega,\epsilon,\mu) + \inf(\mu\wedge r) \end{aligned}$$

$\diamond$

Now we can prove the main result of this section.

Theorem 3.6 *Let $\mathcal{G}$ be a positive game. Then $\mathcal{G}$ has a c-competitive strategy iff $\mathcal{G}^c$ has a memoryless finite-cost strategy.*

Proof: ($\Leftarrow$) Suppose $\mathcal{A}^c$ is a finite-cost strategy for $\mathcal{G}^c$. We define (uniquely) a strategy $\mathcal{A}$ for $\mathcal{G}$ by the equation

$$\mathcal{A}^c((x,\chi_x),\varrho) = (\mathcal{A}(x,\varrho), \chi_x \triangle \varrho).$$

Let $\phi^c : Q \times \mathcal{G}^c \to \mathbf{R}^+$ be the function given by the definition of finite-cost strategy, and let $\phi(x) = \phi^c(x, \chi_x)$. It is a simple matter of checking the definitions to verify that

$$cost_{\mathcal{A}}(x,\varrho) \leq c \cdot cost_{opt}(x,\varrho) + \phi(x).$$

($\Rightarrow$) By the c-competitiveness of $\mathcal{A}$ we have that

$$\lambda(x) = \sup_{\varrho \in \mathcal{R}^*} \{cost_{\mathcal{A}}(x, \varrho) - c \cdot cost_{opt}(x, \varrho)\}$$

is well defined. Define now

$$\psi(x, \omega) = \inf_{\mathcal{A}(z,\varrho)=x} \{\lambda(z) + c \cdot f^0(\chi_z, \varrho, \omega) - cost_{\mathcal{A}}(z, \varrho)\}.$$

Again, ψ is well defined and non-negative since

$$\begin{aligned} \lambda(z) + c \cdot f^0(\chi_z, \varrho, \omega) - cost_{\mathcal{A}}(z, \varrho) &\geq c \cdot \Big[f^0(\chi_z, \varrho, \omega) - cost_{opt}(z, \varrho)\Big] \\ &\geq c \cdot \Big[f^0(\chi_z, \varrho, \chi_z \triangle \varrho) - cost_{opt}(z, \varrho)\Big] \quad = \quad 0. \end{aligned}$$

Fix $x \in Q$, $\omega \in \mathbf{W}^0$ and a request $r \in \mathcal{R}$. Pick z, ϱ that almost realizes the infimum in the definition of $\psi(x, \omega)$, that is $\mathcal{A}(z, \varrho) = x$ and

$$\psi(x, \omega) < \lambda(z) + c \cdot f^0(\chi_z, \varrho, \omega) - cost_{\mathcal{A}}(z, \varrho) + \epsilon$$

for small $\epsilon > 0$. Let $y = \mathcal{A}(z, \varrho r)$. Then, using Lemma 3.2, we have

$$\begin{aligned} \psi(y, \omega \triangle r) &\leq \lambda(z) + c \cdot f^0(\chi_z, \varrho r, \omega \triangle r) - cost_{\mathcal{A}}(z, \varrho r) + \epsilon \\ &\leq \lambda(z) + c \cdot \Big[f^0(\chi_z, \varrho, \omega) + \inf(\omega \wedge r)\Big] - [cost_{\mathcal{A}}(z, \varrho) + f(x, r, y)] + \epsilon \\ &= \Big[\lambda(z) + c \cdot f^0(\chi_z, \varrho, \omega) - cost_{\mathcal{A}}(z, \varrho)\Big] - [f(x, r, y) - c \cdot \inf(\omega \wedge r)] + \epsilon \\ &= \psi(x, \omega) - f^c[(x, \omega), r, (y, \omega \triangle r)] + \epsilon. \end{aligned}$$

Letting $\epsilon \to 0$, we see that ϕ satisfies the hypothesis Theorem 3.2.

This would complete the proof, except for the finiteness hypothesis of Theorem 3.2 – unfortunately, $\mathcal{G}^c$ has infinitely many states. But we can use the finiteness of Q itself to get around the problem.[5] Instead of using Theorem 3.2, we explicitly construct the algorithm, that is we define the state $(y, \omega') = \mathcal{A}'((x, \omega), r)$ that will be chosen by the memoryless algorithm $\mathcal{A}'$ in $\mathcal{G}^c$. For each $\epsilon > 0$, let $y_\epsilon \in Q$ be the value of y chosen in the above argument. Since Q is finite, we can pick $y \in Q$ such that

$$\forall \delta > 0 \ : \ \exists\, 0 < \epsilon < \delta \ : \ y = y_\epsilon.$$

Then $\mathcal{A}'((x, \omega), r) = (y, \omega \triangle r)$ is the state we move to if the request is r. $\diamond$

[5]This problem will not appear when we allow infinite sets of states with appropriate topological structure. This will be done in the final version of this paper.

3.2.2 Example: the List-Update Problem for Two Items

A simple, yet interesting, example of an on-line problem is the list update problem for a list of length 2 (LUP2), with paid exchanges. Let a, b be the items that are in the list. The list can be in two states: $x = ab$ and $y = ba$, and the request set is $\{a, b\}$. The adversary requests an item (either a or b). We service the request by paying i if the requested item is i^{th} in the list. We can also exchange the two items at any time at a cost of 1.

There are three accessible offset functions, that we denote 00, 01 and 10. (We call an offset function *accessible* if it is of the form $\chi_u \triangle \varrho$, for some state u and request sequence ϱ). Function ij has value i on x and j on y.

Thus the on-line game $\mathcal{G}^{4/3}$ for LUP2 has $2 \cdot 3 = 6$ states, as illustrated in Figure 9. Only essential transitions are shown (*i.e.*, transitions between states (z, ij), $(t, k\ell)$ where $k\ell = ij \triangle r$ for some $r \in \{a, b\}$). A memoryless finite cost algorithm for $\mathcal{G}^{4/3}$ is described with bold arcs.

3.3 Randomized Strategies for On-Line Games

In this section we consider randomized strategies. The results mentioned here are very preliminary, and are included only to outline the direction of our research. Our purpose is to show results similar to those for the deterministic case: that the problem of finding a competitive randomize algorithm for a positive game can be reduced to finding a memoryless finite cost algorithm for some other game.

Let $\mathcal{G} = \langle Q, \mathcal{R}, f \rangle$ be a positive on-line game. A randomized strategy $\mathcal{A}$ is a function $\mathcal{A} : Q \times \mathcal{R}^* \to \Pi_Q$, where Π_Q is the set of all probability distributions on Q.[6]

$\mathcal{A}$ is called *stable* if there exists a function $\mathbf{W}^0 \to \Pi_Q$ (write π_ω for the distribution associated with ω under this function) such that $\mathcal{A}(x, \varrho) = \pi_{x \triangle \varrho}$ for any non-empty string of requests ϱ. In other words, the probability distribution of $\mathcal{A}$ depends only on the current offset function.

The following hypothesis, although not universal, is very useful in designing randomized algorithms for on-line problems. We know that this hypothesis is not true for *all* on-line games. However, it has turned out to be true for all natural on-line problems that we have encountered so far. We believe that it should hold under some natural assumptions that we have not yet been able to determine.

> **Stable Distribution Hypothesis (SDH):** If a positive game $\mathcal{G}$ has a randomized c-competitive strategy, then it also has a stable randomized c-competitive strategy.

[6] This definition is correct only if Q is finite. Using well-known probability theory constructions, the general case will be developed similarly in the published version.

Let now $\mathcal{G} = \langle Q, \mathcal{R}, f \rangle$ be an arbitrary game. Given two probability distributions $\pi, \theta \in \Pi$, and $r \in \mathcal{R}$, we define $f_\Pi(\pi, r, \theta)$ to be the "minimum transport" distance between π and θ. Formally, if $Q = \{x^1, \ldots, x^n\}$, $f_\Pi(\pi, r, \theta)$ is the minimum value of

$$\sum_{1 \le i,j \le n} \pi(x^i)\theta(y^j) f(x^i, r, x^j) a_{ij}$$

subject to the constraints

$$\begin{aligned} \forall i, j \ : \ a_{ij} &\ge 0 \\ \forall i \ : \ \sum_j a_{ij} &= \pi(x^i) \\ \forall j \ : \ \sum_i a_{ij} &= \theta(y^j) \end{aligned}$$

Define another game $\mathcal{G}_\Pi = \langle \Pi, \mathcal{R}, f_\Pi \rangle$.[7]

Theorem 3.7 *$\mathcal{G}$ has a finite-cost memoryless randomized strategy iff $\mathcal{G}_\Pi$ has a finite-cost memoryless deterministic strategy.*

Proof: Omitted. $\diamond$

Using Theorem 3.7, and ideas similar to those from the previous section, it should be possible to show that the existence of a c-competitive randomized algorithm in a positive game $\mathcal{G}$ is equivalent to the existence of a finite-cost deterministic algorithm in some different game $\mathcal{G}'$. We have not yet worked out all the details of this reduction in terms of on-line games. This reduction, together with SDH, yields a useful technique for finding randomized algorithms for on-line problems. We have already applied this technique successfully in some problems we have worked on. The benefit from using SDH is that it reduces the complexity of $\mathcal{G}'$, by either reducing the number of states in the case of finite games, or simplifying the description of $\mathcal{G}'$ if $\mathcal{G}$ is infinite.

3.3.1 Example: the Randomized List-Update Problem for Two Items

Let RLUP2 be the randomized version of LUP2, presented in 3.2.2. A stable 9/8-competitive randomized strategy is given by the distributions $\pi_{01}, \pi_{00}, \pi_{10}$ where

$$\begin{array}{llllllllll} \pi_{01}(x) & = & 7/8 & \pi_{01}(y) & = & 1/8 & \pi_{00}(x) & = & 1/2 \\ \pi_{00}(y) & = & 1/2 & \pi_{10}(x) & = & 1/8 & \pi_{10}(y) & = & 7/8 \end{array}$$

There is a matching lower bound, as well. If the adversary selects an input sequence of length N at random, for large N, the expected cost of any randomized algorithm will be 9/8 as great as the adversary's cost.

[7] Of course, $\mathcal{G}_\Pi$ is not finite. But it certainly satisfies the topological conditions that we will need to extend our results, in the published version.

In a way, this result is counter-intuitive. "Common sense" would seem to indicate that our distribution should eventually be concentrated at x if the request sequence contains a sufficiently long string of a's, but that is not the case – it can be shown that the randomized strategy given above is the unique optimal one.

The lower bound was first found by Karp and Raghavan.[8]

4 Metrical Service Systems

A *Metrical Service System* is pair $\mathcal{S} = \langle M, \mathcal{R} \rangle$, where M is a metric space and $\mathcal{R} \subseteq 2^M$ is a family of subsets of M. Each $r \in \mathcal{R}$ is called a *request*.

We are given one server that can occupy and move among the points of M. At every time step a request $r \in \mathcal{R}$ appears, and we need to move the server to some point $x \in r$ to "serve" the request r. Our cost at this step is equal to the distance from the previous position of the server to x.

Our results. We present several preliminary results about metrical service systems:

1. We prove the lower bound of $2m - 1$ for the competitiveness constant if $\mathcal{R}$ contains sets of size m.

2. We consider the case of m-point requests, in a uniform metric space. For this problem we give an optimal deterministic m-competitive algorithm, and also an optimal randomized H_m-competitive algorithm.

3. Then we consider the case when $m = 2$. For this case we present a 9-competitive algorithm and we conjecture that this constant is optimal. There is also a c-competitive randomized algorithm for this problem (also conjectured to be optimal), for $c = 4.59112\ldots$, which we will present in the published version.

4.1 A Lower Bound for m-Point Requests

Denote by MSS_m the class of metrical service systems where all request sets have cardinality at most m.

Theorem 4.1 *There are no c-competitive algorithms for* MSS_m *if* $c < 2m - 1$.

Proof: We describe a sequence of m moves which the adversary can execute at cost 1, which forces any algorithm to pay $2m - 1$. The adversary can execute this sequence whenever its server matches the algorithm's server, and, at the end of the sequence, the servers are again matched. By iterating this sequence, the adversary can defeat any algorithm which is

[8] Personal Communication.

supposedly less than $(2m-1)$-competitive.

Initially s is on point x. Let r^0 be any set of m points such that $yx = 1$ and $yz = 2$ for all $y, z \in r^0$. The algorithm then requests the sequence of sets $r^1, r^2, \ldots r^{m-1}$, characterized by

- r^t has cardinality $m - t$.
- r^t never contains our server's position.
- $r^t \subseteq r^{t-1}$ for $0 < t < m$.

Let $r^{m-1} = \{x'\}$. The algorithm must move its server at each step, and thus incurs a total cost of $2m - 1$. The adversary, however, simply decides to move its server from x to x', at a cost of 1. $\diamond$

4.2 Uniform Metric Spaces

A metric space is uniform if all distances are 1. In this section we present a deterministic m-competitive algorithm and a H_m-competitive randomized algorithm for uniform metric spaces. Both those constants are best possible.

Theorem 4.2 *There is an m-competitive algorithm for* MSS_m *on uniform metric spaces, and the constant m is optimal.*

Proof: The upper bound. $\mathcal{R}$ is now the set of all subsets of the uniform space that contain at most m points. Let S_ω be the set of points $x \in M$ such that $\omega(x) = 0$, where ω is an offset function. We call S_ω the *support* of ω.[9] $S_\omega \subseteq r$, where r is the last request.

If r is a new request, then

$$S_{\omega \triangle r} = \begin{cases} r & \text{if } r \cap S_\omega = \emptyset \\ S_\omega \cap r & \text{otherwise} \end{cases}$$

Our algorithm $\mathcal{A}$ is as follows: Let s be the current location of the server and r the new request. If $s \in r$, stay on s. Otherwise, move to any point in $S_{\omega \triangle r}$. Notice that the server will always move to a point of offset 0.

We define the potential function $\Phi(x, \omega) = |S_\omega|$. We need to show that at every move we have

$$\Delta cost_{\mathcal{A}} + |S_{\omega \triangle r}| - |S_\omega| \;\leq\; m \cdot \inf(\omega \wedge r). \tag{5}$$

We have three cases.

[9] This is consistent with the definition of support in 2.3.

(a) $S_{\omega\triangle r} \subseteq S_\omega$. Then $\inf(\omega \wedge r) = 0$.

(a1) $s \in S_{\omega\triangle r}$. Then our cost is 0, and inequality (5) is obvious.

(a2) $s \notin S_{\omega\triangle r}$. Then we must have $|S_{\omega\triangle r}| < |S_\omega|$, implying (5).

(b) $S_{\omega\triangle r} \cap S_\omega = \emptyset$. Then $\inf(\omega\triangle r) = 1$. Since $|S_\omega| \geq 1$ and $|S_{\omega\triangle r}| = m$, inequality (5) is true in this case, too.

The lower bound. Assume that M has at least $m+1$ points. The adversary first requests r^0, with $|r^0| = m$, such that $s \notin r^0$. At step $t > 0$, the adversary selects $r^t = r^{t-1} - \{s\}$, where s is the position of our server after the previous step. Finally, $|r^{m-1}| = \{y\}$ is a singleton set. The adversary decides now to serve all the requests r^t from y. The adversary's cost is 1, and our cost is m. $\diamond$

Theorem 4.3 *Let H_m denote the m-th harmonic number. There is a H_m-competitive randomized algorithm for* MSS_m *on uniform metric spaces, and the constant H_m is optimal.*

Proof: The upper bound. We use the notation from Theorem 4.2. Additionally, let $i_\omega = |S_\omega|$. The strategy we give is stable (see 3.3), and is defined by

$$\pi_\omega(x) = \begin{cases} 1/i_\omega & \text{if } x \in S_\omega \\ 0 & \text{otherwise} \end{cases}$$

We define the potential function by $\Phi(\omega) = H_{i_\omega}$. We need to show that for each ω and r we have

$$\pi_\omega \pi_{\omega\triangle r} + \Phi(\omega\triangle r) \leq \Phi(\omega) + H_m \cdot \inf(\omega \wedge r).$$

(Recall that $\pi_\omega\pi_{\omega\triangle r}$ is the "minimum transport" distance.)

We distinguish two cases.

(1) $S_{\omega\triangle r} \subseteq S_\omega$. Then $\inf(\omega\wedge r) = 0$, and

$$\begin{aligned} \pi_\omega \pi_{\omega\triangle r} + \Phi(\omega\triangle r) &= \frac{i_\omega - i_{\omega\triangle r}}{i_\omega} + H_{i_{\omega\triangle r}} \\ &\leq \frac{1}{i_{\omega\triangle r}+1} + \cdots + \frac{1}{i_\omega} + H_{i_{\omega\triangle r}} \\ &= H_{i_\omega} = \Phi(\omega). \end{aligned}$$

(2) $S_{\omega\triangle r} \cap S_\omega = \emptyset$. Then $\inf(\omega\wedge r) = 1$, and $\pi_\omega\pi_{\omega\triangle r} = 1$. Therefore

$$\pi_\omega \pi_{\omega\triangle r} + \Phi(\omega\triangle r) = 1 + H_m \leq H_{i_\omega} + H_m.$$

This completes the proof of the upper bound.

Lower bound. Assume that M has at least $m+1$ points. We give a cycle where the adversary cost is 1 and the expected cost of any randomized algorithm is at least H_m.

Suppose that the distribution of our server is concentrated on one point, s. The adversary picks a sequence of requests, $r^0, r^1, \ldots r^{m-1}$. Let π^t be the distribution of our server after we serve the request r^t. The adversary's strategy is to choose r^0 to be any set of m points that does not include s. For $0 \leq t < m-1$, he then chooses a point $x^t \in r^t$ such that $\pi^{t-1}(x^{t-1}) \geq \frac{1}{m-t}$, and lets $r^{t+1} = r^t - \{x\}$. Finally, r^{m-1} contains a single point y, and the distribution of our server is again concentrated at a single point. The adversary chooses to server all the requests at y, and thus pays 1, while our expected cost is at least $1 + \frac{1}{m} + \ldots + \frac{1}{2} = H_m$. $\diamond$

4.3 Two-Point Request Sets

Theorem 4.4 *There is a 9-competitive algorithm for* MSS_2.

Proof: (Sketch) We only show here the potential function Φ for which the generic algorithm (as defined in the section about on-line games) is 9-competitive. Suppose that $r = \{x, y\}$ is the last request, and $s = x$ is the current position of our server. The number $\omega = \omega(y)$ describes uniquely the current offset function since only one offset in the support is non-zero. Let also $d = xy$ be the distance between the current request points. In this notation the potential function defined to be

$$\Phi = \begin{cases} 0 & \text{if } \omega \geq d/3 \\ 2d - 6\omega & \text{if } -d/3 \leq \omega \leq d/3 \\ 4d & \text{if } \omega \leq -d/3 \end{cases}$$

$\diamond$

Conjecture 4.1 MSS_2 *is no better than* 9*-competitive.*

Theorem 4.5 *There is a c-competitive randomized algorithm for* MSS_2*, where c is the solution of the equation* $ln(c-1) = \frac{c}{c-1}$*, approximately* 4.59112.

Proof: Omitted.$\diamond$

Conjecture 4.2 *Randomized* MSS_2 *is no better than c-competitive, where c is the solution of the equation* $ln(c-1) = \frac{c}{c-1}$.

5 Final Comments

We believe that essential progress in the area of on-line algorithms will not be possible without sound mathematical foundations. Neither it can progress without developing general tech-

niques for developing on-line algorithms.

We have presented two formalizations of on-line problems: on-line games and Metrical Service Systems. In terms of on-line games we explained the usefulness of the potential method, and showed that the problem of finding on-line competitive algorithms can be reduced to the problem of finding memoryless finite-cost algorithms. Thus the latter problem, somewhat simpler than the competitive analysis, captures the difficulty of the problem.

Metrical Service Systems contain a number of problems that are both natural and mathematically interesting. We have given a number of examples and developed some competitive algorithms for them.

The General Work Function Algorithm. In Section 2.3 we introduced for the k-server problem. The concept generalizes naturally to a wide collection of on-line problems. We use the terminology from on-line games.

The Greedy algorithm is to always move to a state as to minimize the current cost. An alternative strategy would be to compute (using, say, dynamic programming) what state the adversary is in, assuming that he has played optimally and this is the last request, then move to that state. This could have great cost, as we could be in a state quite far from that one. Such a strategy we could call the "Chase-Optimal" strategy.

The General Work Function Algorithm is intermediate between these two extremes. It has a non-negative parameter λ. If x^0 is the initial state, and x is the current state of the algorithm, then the General Work Function Algorithm, when it receives input r, chooses to move to that state x' which minimizes

$$cost_{WFA}(x, r, x') + \lambda \cdot cost_{opt}(x^0, \varrho, x')$$

where ϱ is the sequence of inputs so far. Note that the Greedy algorithm is a special case of the work function algorithm obtained by letting $\lambda = 0$, while the Chase-Optimal algorithm is obtained by letting $\lambda = \infty$.

The Work Function Algorithm introduced in 2.3 uses $\lambda = 1$. We have discovered that the general work function algorithm has other applications, as well. For example, in [6], we introduced an algorithm which is just the work function algorithm for $\lambda = 1$ for a simplified version of the server problem.

Forgiveness. The Equipoise algorithm uses a technique which is hinted at in the construction of the game $\mathcal{G}^c$ introduced in 3.2. We call this technique *forgiveness*.

As mentioned during the discussion of $\mathcal{G}^c$, it is pointless for the algorithm to pick a transition $(x, \omega) \to (y, \mu)$ unless $\mu = \omega \triangle r$. The reason is that, if it does, it is "discarding." or "forgiving" a portion of the adversary's obligation, without making him pay for it. But, this is exactly what Equipoise does – it "forgives to a cone."

The purpose of forgiving a portion of the adversary's obligation is to simplify the algorithm. In fact, Equipoise can be considered to be a certain strategy for the game $\mathcal{G}^c$ (where $\mathcal{G}$ is the game corresponding to the k-server problem) where only states of the form (x, χ_x) are used. The value of such a strategy is that the subgame of $\mathcal{G}^c$ considered only has as many states as $\mathcal{G}$ itself, simplifying the analysis, at a possible loss of optimality. Equipoise can be thought of as an answer to the question, what is a good strategy for this subgame?[10]

Nevertheless, Equipoise is optimal for $k = 2$, in spite of its forgiving nature. We suspect that for higher k, some forgiveness can be allowed (though not all the way to a cone) without sacrificing optimality. We are working on this idea.

References

[1] S. Ben-Davida, A. Borodin, R. Karp, G. Tardos, and A. Widgerson. On the power of randomization in on-line algorithms. In *Proc. 22nd Symposium on Theory of Algorithms*, pages 379–386, 1990.

[2] A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. In *Proc. 19th Annual ACM Symposium on Theory of Computing*, pages 373–382, 1987.

[3] M. Chrobak, H. Karloff, T. Payne, and S. Vishwanathan. New results on server problems. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 291–300, 1990.

[4] M. Chrobak and L. Larmore. Harmonic is three-competitive for two servers. Submitted for publication.

[5] M. Chrobak and L. Larmore. A new approach to the server problem. *SIAM Journal on Discrete Mathematics*, 1991. To appear.

[6] M. Chrobak and L. Larmore. A note on the server problem and a benevolent adversary. *Information Processing Letters*, 1991. To appear.

[7] M. Chrobak and L. Larmore. On fast algorithms for two servers. *Journal of Algorithms*, 1991. To appear.

[8] M. Chrobak and L. Larmore. An optimal online algorithm for k servers on trees. *SIAM Journal on Computing*, 20:144–148, 1991.

[9] D. Coppersmith, P. G. Doyle, P. Raghavan, and M. Snir. Random walks on weighted graphs and applications to online algorithms. In *Proc. 22nd Annual ACM Symposium on Theory of Computing*, pages 369–378, 1990.

[10] It was out of analysis of Equipoise that work functions were conceived, however, not the other way around.

[10] A. Fiat, R. Karp, M. Luby, L.A. McGeoch, D. Sleator, and N.E. Young. *Competitive paging algorithms.* Technical Report CMU-CS-88-196, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, 1988.

[11] A. Fiat, Y. Rabani, and Y. Ravid. Competitive k-server algorithms. In *Proc. 22nd Symposium on Theory of Algorithms*, 1990.

[12] E. Grove. The harmonic on-line k-server algorithm is competitive. Manuscript.

[13] S. Irani and R. Rubinfeld. A competitive 2-server algorithm. Manuscript.

[14] M. Manasse, L. A. McGeoch, and D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.

[15] M. Manasse, L.A. McGeoch, and D. Sleator. Competitive algorithms for server problems. In *Proc. 20th Annual ACM Symposium on Theory of Computing*, pages 322–333, 1988.

[16] L. McGeoch and D. Sleator. A strongly competitive randomized paging algorithm. Manuscript.

[17] P. Raghavan and M. Snir. Memory versus randomization in online algorithms. In *16th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Computer Science vol. 372*, pages 687–703, Springer-Verlag, 1989.

[18] D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communnications of ACM*, 28:202–208, 1985.

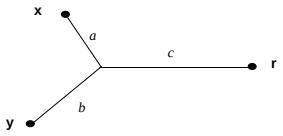

Figure 1: Convex hull of three points x, y, r.

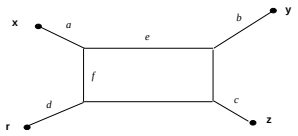

Figure 2: Convex hull of four points x, y, z, r.

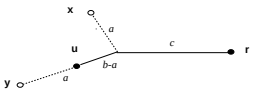

Figure 3: Equipoise moves servers from $\{x, y\}$ to $\{r, u\}$.

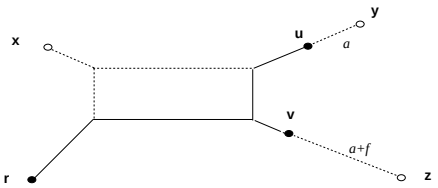

Figure 4: Equipoise's move in Case I.

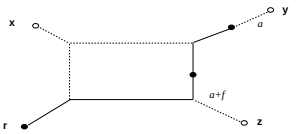

Figure 5: Equipoise's move in Case II.

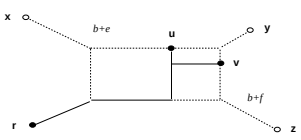

Figure 6: Equipoise's move in Case III.

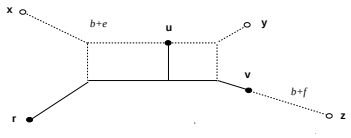

Figure 7: Equipoise's move in Case IV.

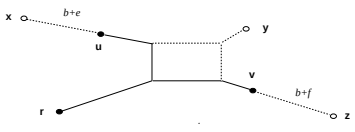

Figure 8: Equipoise's move in Case V.

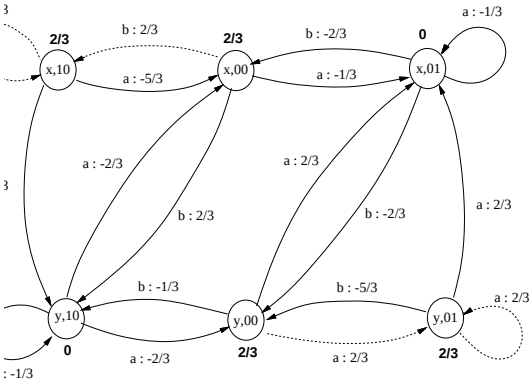

Figure 9: The game $\mathcal{G}^{4/3}$ for the list-update problem with two items. Notation "a : 2/3" means that the cost of the given transition on request a is 2/3. Bold-face numerals represent potentials.