Algorithms **2008**, *1*, 30-42; DOI: 10.3390/a1010030

OPEN ACCESS

algorithms

ISSN 1999-4893

www.mdpi.org/algorithms

Article

Randomized Competitive Analysis for Two Server Problems

Wolfgang Bein [1,⋆,†], Kazuo Iwama [2] and Jun Kawahara [2]

[1] Center for the Advanced Study of Algorithms, School of Computer Science, University of Nevada, Las Vegas, Nevada 89154, USA

[2] School of Informatics, Kyoto University, Kyoto 606-8501, Japan

Email addresses: bein@cs.unlv.edu; iwama@kuis.kyoto-u.ac.jp; jkawahara@kuis.kyoto-u.ac.jp

⋆ Author to whom correspondence should be addressed.

† Research done while visiting Kyoto University as Kyoto University Visiting Professor.

Received: 13 August 2008 / Accepted: 18 September 2008 / Published: 19 September 2008

Abstract: We prove that there exists a randomized online algorithm for the 2-server 3-point problem whose expected competitive ratio is at most 1.5897. This is the first nontrivial upper bound for randomized k-server algorithms in a general metric space whose competitive ratio is well below the corresponding deterministic lower bound (= 2 in the 2-server case).

Keywords: Randomized Algorithms; Online Algorithms; Server Problems.

---

1. Introduction

The k-server problem, introduced by Manasse, McGeoch and Sleator [1], is one of the most fundamental online problems. In this problem the input is given as k initial server positions and a sequence $p_1, p_2, \cdots$ of requests in the Euclidean space, or more generally in any metric space. For each request p_i, the online player has to select, without any knowledge of future requests, one of the k servers and move it to p_i. The goal is to minimize the total moving distance of the servers.

The k-server problem is widely considered instructive to the understanding of online problems in general, yet, there are only scattered results. The most notable open problem is perhaps the k-server conjecture, which states that the k-server problem is k-competitive. The conjecture remains open for $k \geq 3$, despite years of effort by many researchers; it is solved for a very few special cases, and remains open even for 3 servers when the metric space has more than 6-points. The current best upper bound is $2k - 1$ given by Koutsoupias and Papadimitriou in 1994 [2]. The conjecture is true for $k = 2$, for the

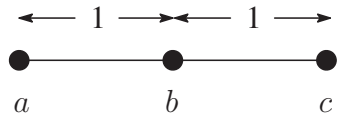

Figure 1. 3 points on a line

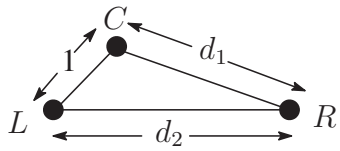

Figure 2. Triangle CLR

line [3], trees [4], and on fixed $k+1$ or $k+2$ points [5]. It is still open for the 3-server problem on more than six points and also on the circle [6]. The lower bound is k which is shown in the original paper [1].

In the randomized case, even less is known. Indeed, one of hardest problems in the area of online algorithms is to determine the exact randomized competitiveness of the k-server problem, that is, the minimum competitiveness of any randomized online algorithm for the server problem. (As is customary, we mean by "competitive ratio" of a randomized algorithm its expected compete ratio.) Very little is known for general k. Bartal et al. [7] have an asymptotic lower bound, namely that the competitiveness of any randomized online algorithm for an arbitrary metric space is $\Omega(\log k/\log^2 \log k)$.

Even the case $k = 2$ is open for the randomized 2-server problem, and, despite much effort, no randomized algorithm for general metric spaces with competitiveness strictly lower than 2 has been found. Surprisingly, the classic algorithm RANDOM SLACK [8], a very simple trackless algorithm, has been the algorithm with best competitive ratio for almost two decades now. Karlin et al. [9] gave a lower bound of $\frac{e}{e-1}$, which is the bound for the classical "ski rental problem". This bound is derived in a space with three points, where two points are located closely together and the third point is far from both of these. The best known lower bound is $1 + e^{-\frac{1}{2}} \approx 1.6065$, but this lower bound requires a space with at least four points, see [10]. (A lower bound very slightly larger than $1 + e^{-\frac{1}{2}}$ is stated in [10], but without proof.)

It is indeed surprising that no randomized algorithm with competitive ratio better than 2 has been found, since it seems intuitive that randomization should help. It should be noted that generally randomization is quite powerful for online problems, since it obviously reduces the power of the adversary. Such seems to be the case for the 2-server problem as well.

To give intuition, consider a simple 2-server problem on the three equally spaced points a, b and c on a line (See Fig. 1). It is easy to prove a lower bound of 2 for the competitive ratio of any deterministic algorithm: The adversary always gives a request on the point the server is missing. Thus for any online algorithm, $\mathcal{A}$, its total cost is at least n – the number of requests. But it turns out by a simple case analysis that the offline cost is $n/2$.

Suppose instead that $\mathcal{A}$ is randomized. Now if the request comes on b (with missing server), then $\mathcal{A}$ can decide by a coin flip which server (a or c) to move. An (oblivious) adversary knows $\mathcal{A}$'s algorithm completely but does not know the result of the coin flip and hence cannot determine which point (a or c) has the server missing in the next step. The adversary would make the next request on a but this time a has a server with probability $1/2$ and $\mathcal{A}$ can reduce its cost. Without giving details, it is not hard to show that this algorithm $\mathcal{A}$ – with the randomized action for a request to b and a greedy action one for others – has a competitive ratio of 1.5.

Indeed, one would imagine that it might be quite straightforward to design randomized algorithms which perform significantly better than deterministic ones for the 2-server problem. As mentioned above

this has not been the case. Only few special cases have yielded success. Bartal, Chrobak, and Larmore gave a randomized algorithm for the 2-server problem on the line, whose competitive ratio is slightly better than 2 ($\frac{155}{78} \approx 1.987$) [11]. One other result by Bein et. al. [12] uses a novel technique, the knowledge state method, to derive a $\frac{19}{12}$ competitive randomized algorithm for the special case of Cross Polytope Spaces. Using similar techniques a new result for paging (the k-server problem in uniform spaces) was recently obtained. Bein et al. [13] gave an H_k-competitive randomized algorithm which requires only $O(k)$ memory for k-paging. (Though the techniques in the current paper are inspired by this work, the knowledge state method is not used here.) Lund and Reingold showed that if specific three positions are given, then an optimal randomized algorithm for the 2-server problem over those three points can be derived in principle by using linear programming [14]. However, they do not give actual values of its competitive ratio and to this date the problem is still open even for the 2-server 3-points case.

Finally, we mention other somewhat related work in the realm of online computation: see the work of Karlin et al. [15] for ski-rental problems, Reingold et al. [16] for list access problems, and Fiat et al. [17] for paging.

Our Contribution. In this paper, we prove that the randomized competitive ratio of the 2-server 3-point problem in a general metric space is at most 1.5897 and also we conjecture that it is at most $e/(e-1) \approx 1.5819$. Thus we give an upper bound that matches the lower bound within a small ε.

The underlying idea is to find a finite set S of triangles (i.e. three points) such that if the competitive ratio for each triangle in S is at most c, then the competitive ratio for all triangles in any metric space is at most $c \cdot \delta(S)$ where $\delta(S) \geq 1$ is a value determined by S. To bound the competitive ratio for each triangle in S, we apply linear programming. As we consider larger sets, the value of $\delta(S)$ becomes smaller and approaches 1. Thus the upper bound of the general competitive ratio also approaches the maximum competitive ratio of triangles in S and we can obtain arbitrarily close upper bounds by increasing the size of the computation.

Our result in this paper strongly depends on computer simulations similar to earlier work based on knowledge states. Indeed, there are several successful examples of such an approach, which usually consists of two stages; (i) reducing infinitely many cases of a mathematical proof to finitely many cases (where this number is still too large for a "standard proof") and (ii) using computer programs to analyze the finitely many cases. See the work in [18–22] for design and analysis of such algorithms. In particular, for online competitive analysis, Seiden proved the currently best upper bound, 1.5889, for online bin-packing [23]. Also with this approach, Horiyama et al. [24] obtained an optimal competitive ratio for the online knapsack problem with resource augmentation by buffer bins.

2. Our Approach

Since we consider only three fixed points, we can assume without loss of generality that they are given in the two-dimensional Euclidean space. The three points are denoted by L, C and R, furthermore let $d(C, L) = 1$, $d(C, R) = d_1$, and $d(L, R) = d_2$ (see Fig. 2). Without loss of generality we assume that $1 \leq d_1 \leq d_2 \leq d_1 + 1$, since the actual lengths of the triangle's sides are irrelevant and only their ratios matter. The 2-server problem on L, C and R is denoted by $\Delta(1, d_1, d_2)$, where the two servers are

on L and R initially and the input is given as a sequence σ of points $\in \{L, C, R\}$. $\Delta(1, d_1, d_2)$ is also used to denote the triangle itself. The cost of an online algorithm $\mathcal{A}$ for the input sequence σ is denoted by $ALG_{\mathcal{A}}(\sigma)$ and the cost of the offline algorithm by $OPT(\sigma)$. Suppose that for some constant $\alpha \geq 0$, $E[ALG_{\mathcal{A}}(\sigma)] \leq r \cdot OPT(\sigma) + \alpha$, holds for any input sequence σ. Then we say that the competitive ratio of $\mathcal{A}$ is at most r.

We first consider the case that the three points are on a line and both d_1 and d_2 are integers. In this case, we can design a general online algorithm as follows. The proof is given in the next section.

Lemma 1 *Let n be a positive integer. Then there exists an online algorithm for $\Delta(1, n, n+1)$ whose competitive ratio is at most $C_n = \frac{(1+\frac{1}{n})^n - \frac{1}{n+1}}{(1+\frac{1}{n})^n - 1}$.*

Note that if triangles Δ_1 and Δ_2 are different, then "good" algorithms for Δ_1 and Δ_2 are also different. However, the next lemma says that if Δ_1 and Δ_2 do not differ too much, then one can use an algorithm for Δ_1 as an algorithm for Δ_2 with small sacrifice on the competitive ratio.

Lemma 2 *Suppose that there are two triangles $\Delta_1 = \Delta(1, a_1, b_1)$ and $\Delta_2 = \Delta(1, a_2, b_2)$ such that $a_1 \geq a_2$ and $b_1 \geq b_2$ and that the competitive ratio of algorithm $\mathcal{A}$ for Δ_1 is at most r. Let $\alpha = \max(\frac{a_1}{a_2}, \frac{b_1}{b_2})$. Then the competitive ratio of $\mathcal{A}$ for Δ_2 is at most $r \cdot \alpha$.*

Proof. For α let $\Delta_\alpha = \Delta(1/\alpha, a_1/\alpha, b_1/\alpha)$. Fix an arbitrary input sequence σ and let the optimal offline cost against σ be OPT_1, OPT_2 and OPT_α for Δ_1, Δ_2 and Δ_α, respectively. Since Δ_α is similar to Δ_1 and the length of each side is $1/\alpha$, OPT_α is obviously $(1/\alpha)OPT_1$. Since every side of Δ_2 is at least as long as the corresponding side of Δ_α, $OPT_2 \geq OPT_\alpha = (1/\alpha)OPT_1$.

Let the expected cost of $\mathcal{A}$ against σ for Δ_1 and Δ_2 be ALG_1 and ALG_2, respectively. Note that $\mathcal{A}$ moves the servers exactly in the same (randomized) way for Δ_1 and Δ_2. Since each side of Δ_2 is at most as long as the corresponding side of Δ_1, $ALG_2 \leq ALG_1$. We have $\frac{ALG_2}{OPT_2} \leq \frac{ALG_1}{(1/\alpha)OPT_1} = \max(\frac{a_1}{a_2}, \frac{b_1}{b_2}) \cdot \frac{ALG_1}{OPT_1}$. □

Thus we can "approximate" all triangles, whose α-value is at most within some constant, by a finite set S of triangles. More precisely we introduce the notion of an approximation set as follows: Suppose that a target competitive ratio, i.e. a competitive ratio one wishes to achieve, is r_0. Then we first calculate the minimum integer n_0 such that $r_0 \geq \frac{n_0+2}{n_0} \cdot C_{n_0+1}$, where C_{n_0+1} is the value given in the statement of Lemma 1. Next we construct a set S such that for any two numbers a and b with $1 \leq a \leq n_0$ and $b \leq a+1$, there exist two triangles $\Delta_1 = \Delta(1, a_1, b_1)$ and $\Delta_2 = \Delta(1, a_2, b_2)$ in S such that the following conditions are met:

(i) $a_2 < a \leq a_1$ and $b_2 < b \leq b_1$,

(ii) there exists an algorithm for Δ_1 whose competitive ratio is r_1, and

(iii) $r_1 \cdot \max(\frac{a_1}{a_2}, \frac{b_1}{b_2}) \leq r_0$.

We call such a set an r_0-*approx* set.

Lemma 3 *If one can construct an r_0-approx set S, then there is an online algorithm for the 2-server problem on three points, whose competitive ratio is at most r_0.*

Proof. Consider the following algorithm $\mathcal{A}(a, b)$ which takes the values a and b of the triangle $\Delta(1, a, b)$. Note that $\mathcal{A}(a, b)$ is an infinite set of different algorithms from which we select one due

to the values of a and b. If $a \geq n_0$, then we select the maximum integer n such that $a \geq n$. Then $\mathcal{A}(a, b)$ uses the algorithm for $\Delta(1, n+1, n+2)$ of Lemma 1. Clearly we have $a \leq n+1$ and $b \leq n+2$. Therefore, by Lemma 2, the competitive ratio of this algorithm for $\Delta(1, a, b)$ is at most (recall that C_{n+1} is the competitive ratio of this algorithm for $\Delta(1, n+1, n+2)$ given in Lemma 1)

$$\max\left(\frac{n+1}{a}, \frac{n+2}{b}\right) \cdot C_{n+1} \leq \frac{n+2}{n} \cdot C_{n+1} \leq \frac{n_0+2}{n_0} \cdot C_{n_0+1} \leq r_0.$$

By a simple calculation we have that $\frac{n+2}{n} \cdot C_{n+1} = \frac{n+2}{n} \cdot \frac{\left(1+\frac{1}{n}\right)^n - \frac{1}{n+1}}{\left(1+\frac{1}{n}\right)^n - 1}$ monotonically decreases, which implies the inequality second to last.

If $a < n_0$, then we have the two triangles Δ_1 and Δ_2 satisfying the conditions (i) to (iii) above. Then we use the algorithm for Δ_1 guaranteed by condition (ii). Its competitive ratio for $\Delta(1, a, b)$ is obviously at most r_0 by Lemma 2. □

3. Three Points on a Line

We now prove Lemma 1. To this end we make use of a state diagram, called the *offset graph*, which indicates the value of the work function $W(s, \sigma)$ [25]. Recall that $W(s, \sigma)$ gives the optimal offline cost under the assumption that all requests given by σ have been served and the final state after σ must be s, where s is one of (L, C), (L, R) and (C, R) in our situation.

Fig. 3 shows the offset graph, G_n^{OPT} for $\Delta(1, n, n+1)$. Each state contains a triple (x, y, z), which represents three values for the work function; the first for when (C, R) are covered (leaving L blank), the next for when (L, R) are covered, and the last for when (L, C) are covered. For instance, in the figure the top middle state is the initial state, which is denoted by V_{LR}. Recall that the initial server configuration is (L, R). This state contains $(n, 0, 1)$, which means that $W((L, C), \phi) = n$, $W((L, R), \phi) = 0$, and $W((C, R), \phi) = 1$. For example, to see that $W((L, C), \phi) = n$, consider that initially the request sequence is empty – denoted by ϕ. The initial server configuration is (L, R); thus in order to change this configuration into (L, C), one can optimally move a server from R to C at cost n. Therefore, $W((L, C), \phi) = n$. Consider now state V_3 – the fourth state from the top. Its triple gives the value of the work function for the request sequence CLC, namely , $W((L, C), CLC)$, $W((L, R), CLC)$ and $W((C, R), CLC)$. Note that this request sequence – CLC – is obtained by concatenating the labels of arrows from the initial state V_{LR} to V_3.

For the work function value of $W((L, R), CLC)$ we have $W((L, R), CLC) = 4$. This is calculated from the previous state V_2 in the following way: Server position (L, R) can be reached from previous configuration (L, R) $(= 2)$ plus 2 (= the cost of moving a server on L to C and back to L) or from previous configuration (C, R) $(= 3)$ plus 1 (= the cost of moving a server on C to L), i.e. in both cases a value of 4. From state V_3, there is an arrow to V_{CR} as a result of request R. Carrying out a similar calculation, one can see that the triple should change from $(n, 4, 3)$ to $(n+4, 4, 3)$ in this transition. However, the triple in V_{CR} is $(n+1, 1, 0)$. This is because of an offset value of 3 on the arrow from V_3 to V_{CR}. Namely, $(n+1, 1, 0)$ in V_{CR} is obtained from $(n+4, 4, 3)$ by reducing each value by 3. Because of the use of offset values a finite graph can be used to represent the potentially infinitely many values of the work function. Thus one can conclude that $(n, 0, 1)$ in the initial state V_{LR} also means

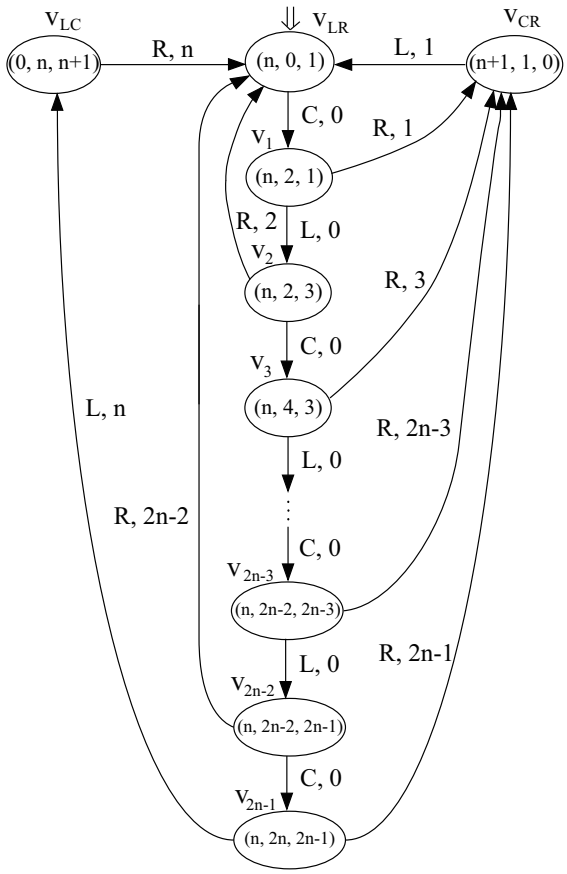

Figure 3. Offset graph

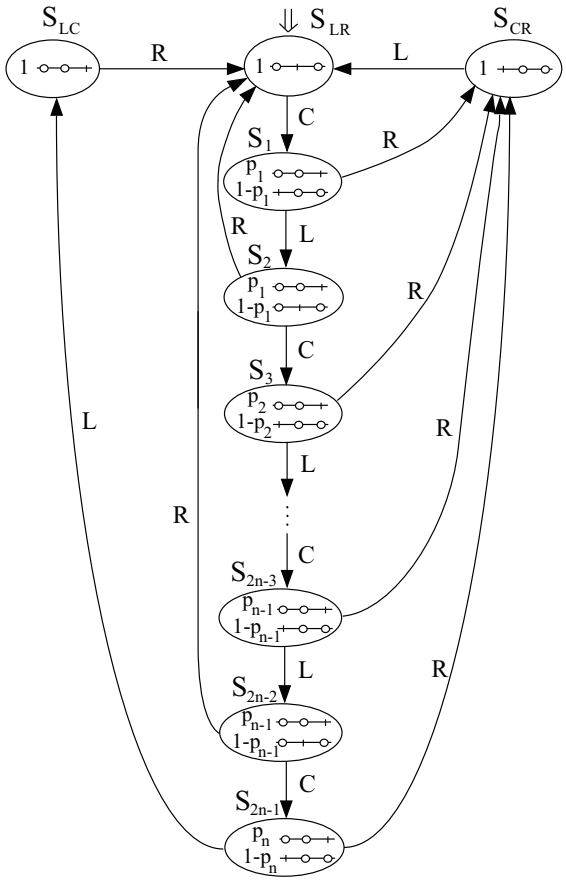

Figure 4. State diagram of the algorithm

$(n+4, 4, 5), (n+8, 8, 9), \cdots$ by traversing the cycle $V_{LR}V_1V_2V_3V_{CR}$ repeatedly. We leave it to the reader to formally verify that Fig. 3 is a valid offset graph for $\Delta(1, n, n+1)$.

We introduce another state graph, called the algorithm graph. Fig. 4 shows the algorithm graph, G_n^{ALG}, for $\Delta(1, n, n+1)$. Notice that G_n^{ALG} is similar to G_n^{OPT}. Each state includes a triple (q_1, q_2, q_3) such that $q_1 \geq 0, q_2 \geq 0, q_3 \geq 0$ and $q_1+q_2+q_3 = 1$, which means that the probabilities of configurations (C, L), (L, R) and (C, R) are q_1, q_2 and q_3, respectively. (Since the most recent request must be served, one of the three values is zero. In the figure, therefore, only two probabilities are given. For example, in S_1, the probabilities for $(L, C)(= p_1)$ and for $(C, R)(= 1 - p_1)$ are given. In our specific algorithm G_n^{ALG}, we define those values as follows:

$$S_{LC} = (1, 0, 0),\ S_{LR} = (0, 1, 0),\ S_{CR} = (0, 0, 1),$$
$$S_{2i-1} = (p_i, 0, 1 - p_i)\ (i = 1, \ldots, n),\ S_{2i} = (p_i, 1 - p_i, 0) \quad (i = 1, \ldots, n-1)$$

where p_i is $\frac{n}{n+1} \cdot \frac{(1+\frac{1}{n})^i - 1}{(1+\frac{1}{n})^n - 1}$.

We describe how to transform an algorithm graph into the actual algorithm. Suppose for example that the request sequence is CL. Assume the algorithm is in state S_2, and suppose that the next request is C. The state transition from S_2 to S_3 occurs. Suppose that S_2 has configuration-probability pairs (C_1, q_1), (C_2, q_2), and (C_3, q_3) ($C_1 = (L, C), C_2 = (L, R)$ and $C_3 = (C, R)$) and S_3 has (C_1, r_1), (C_2, r_2), and (C_3, r_3). We introduce variables x_{ij} $(i, j = 1, 2, 3)$ such that x_{ij} is equal to the probability that the

configuration before the transition is C_i and the configuration after the transition is C_j. Indeed, by a slight abuse of notation the x_{ij} values can be considered to be the algorithm itself. The x_{ij} values also allow us to calculate the cost of the algorithm as described next.

The average cost for a transition is given by $cost = \sum_{i=1}^{3}\sum_{j=1}^{3} x_{ij} d(C_i, C_j)$, where $d(C_i, C_j)$ is the cost to change the configuration from C_i to C_j. We can select the values of x_{ij} in such a way that they minimize the above cost under the condition that $\sum_{j=1}^{3} x_{ij} = q_i, \quad \sum_{i=1}^{3} x_{ij} = r_j$. In the case of three points on the line, it is straightforward to solve this linear program in general. If the servers are on L and C and the request is R, then a greedy move ($C \to R$) is optimal. If the servers are on L and R and the request is C, then the optimal probability is just a proportional distribution due to $d(L, C)$ and $d(C, R)$. The x_{ij} values also show the actual moves of the servers. For example, if the servers are on L and R in S_2, we move a server in L to C with probability x_{23}/q_2 and R to C with probability x_{21}/q_2.

From the values x_{ij}, one can also obtain the expected cost of an algorithm for each transition, as follows:

$\text{cost}(S_{LC}, S_{LR}) = n,\ \text{cost}(S_{CR}, S_{LR}) = 1,\ \text{cost}(S_{LR}, S_1) = np_1 + 1 - p_1,$
$\text{cost}(S_{2i-1}, S_{2i}) = 1 - p_i \quad (i = 1, \ldots, n-1),$
$\text{cost}(S_{2i}, S_{2i+1}) = n(p_{i+1} - p_i) + 1 - p_{i+1} \quad (i = 1, \ldots, n-1),$
$\text{cost}(S_{2i-1}, S_{CR}) = (n+1)p_i \quad (i = 1, \ldots, n),$
$\text{cost}(S_{2i}, S_{LR}) = np_i \quad (i = 1, \ldots, n-1),$
$\text{cost}(S_{2n-1}, S_{LC}) = (n+1)(1 - p_n).$

We are now ready to prove Lemma 1. Recall that G_n^{OPT} and G_n^{ALG} are the same graph. With a request sequence σ, we can thus associate the same sequence, $\lambda(\sigma)$, of transitions in G_n^{OPT} and G_n^{ALG}. The offline cost for $\lambda(\sigma)$ can be calculated from G_n^{OPT} and the online cost from G_n^{ALG}. Considering these two costs, we obtain the competitive ratio for σ.

We need only consider the cycles of the type considered below, as other cycles can be obtained as disjoint unions.

(1) $S_1, S_2, \ldots, S_{2h-1}, S_{CR}, S_{LR} \quad (h = 1, \ldots, n-1)$

(2) $S_1, S_2, \ldots, S_{2h}, S_{LR} \quad (h = 1, \ldots, n-1)$

(3) $S_1, S_2, \ldots, S_{2n-1}, S_{LC}, S_{LR}.$

For sequence (1), the OPT cost is $2h$ and ALG cost is $2np_h + 2h - 2\sum_{j=1}^{h-1} p_j = 2hC_n$. Similarly, for sequence (2), $OPT = 2h$ and $ALG < 2hC_n$ and for sequence (3) $OPT = 2n$ and $ALG = 4n - 2\sum_{j=1}^{n} p_j = 2nC_n$. Thus the competitive ratio is at most C_n for any of these sequences, which proves the lemma. □

4. Construction of a Finite Set of Triangles

For triangle $\Delta_1 = \Delta(1, a, b)$ and $d > 0$, let $\Delta_2 = \Delta(1, a', b')$ be any triangle such that $a - d \le a' \le a$ and $b - d \le b' \le b$. Then as shown in Sec. 2 the competitive ratio for Δ_2, denoted by $f(\Delta_2)$, can be written as

$$f(\Delta_2) \le \max\left(\frac{a}{a'}, \frac{b}{b'}\right) f(\Delta_1) \le \max\left(\frac{a}{a-d}, \frac{b}{b-d}\right) f(\Delta_1) \le \frac{a}{a-d} f(\Delta_1).$$

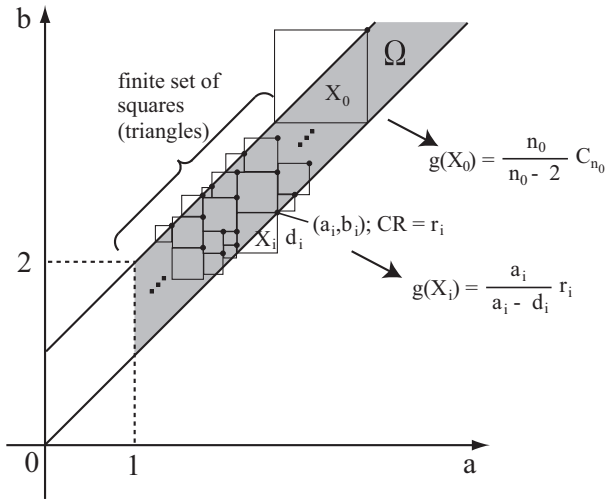

Figure 5. Covering Ω

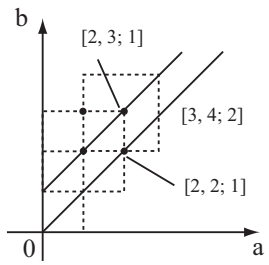

Figure 6. Division of a square

(The last inequality comes from the fact that $a \leq b$.) Recall that triangle $\Delta(1, a, b)$ always satisfies $1 \leq a \leq b \leq a + 1$, which means that (a, b) is in the shaded area of Fig. 5, which we denote by Ω. Consider point (a_i, b_i) in this area and the square X_i of size d_i, whose right upper corner is (a_i, b_i) (Fig. 5). Such a square is also denoted by $[a_i, b_i; d_i]$. Then for any triangle whose (a_i, b_i)-values are within this square (some portion of it may be outside Ω), its competitive ratio can be bounded by $\frac{a_i}{a_i - d_i} f(\Delta(1, a_i, b_i))$, which we call the competitive ratio of the square X_i and denote by $g(X_i)$ or $g([a_i, b_i; d_i])$. Additionally, for $i = 0$ the value $g(X_0)$ is bounded by $\frac{n_0}{n_0 - 2} C_{n_0}$.

Consider a finite set of squares $X_0, X_1, \ldots, X_k = [a_k, b_k; d_k], \ldots, X_m$ with the following properties (see also Fig. 5):

(1) The right-upper corners of all the squares are in Ω.

(2) X_0 is the rightmost square, which must be $[i, i + 1; 2]$ for some i.

(3) The area of Ω between $a = 1$ and i must be covered by those squares, or any point (a, b) in Ω such that $1 \leq a \leq i$ must be in some square.

Suppose that all the values of $g(X_i)$ for $0 \leq i \leq m$ are at most r_0. Then one can easily see that the set $S = \bigcup_{i=0,m} \{\Delta(1, a_i, b_i), \Delta(1, a_i - d_i, b_i - d_i)\}$ of triangles satisfies conditions (i) to (iii) given in Sec. 2, i.e., we have obtained the algorithm whose competitive ratio is at most r_0.

The issue is how to generate those squares efficiently. Note that $g(X)$ decreases if the size d of the square X decreases. Specifically, we can subdivide each square into squares of smaller size to obtain an improved competitive ratio. However, it is not the best policy to subdivide all squares evenly since $g(X)$ for a square X of the same size substantially differs in different positions within Ω. Note this phenomenon especially between positions close to the origin (i.e., both a and b are small) and those far from the origin (the former is larger). Thus our approach is to subdivide squares X dynamically, or to divide the one with the largest $g(X)$ value in each step. Also observe that as squares move to the right in Fig. 5 then the g value decreases due to Lemma 1.

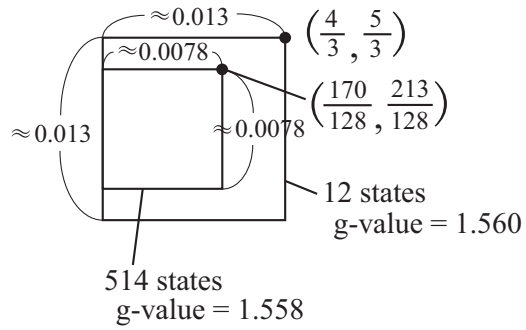

Figure 7. Approximation of a square

We give now an informal description of the procedure for generating the squares, the formal description is given in Procedures 1 and 2. Broadly speaking one starts with square near the origin, then subdivides this square, and continues by creating a new square to the right. Due to Lemma 1 this process is bounded. The procedure begins with a single square $[2, 3; 2]$. Note that its g-value is poor (in fact, not bounded). Next, square $[2, 3; 2]$ is split into four squares of size 1 as shown in Fig. 6: $[1, 3; 1], [1, 2; 1], [2, 2; 1]$ and $[2, 3; 1]$. For each of these squares, the g-value is then calculated in the following way: If the square is of the form $[i, i + 1; \ell]$ for some i, ℓ, the g-value can be calculated immediately by Lemma 1. Otherwise, we use the linear program described in [14] to determine the competitive ratio of triangle $\Delta(1, a, b)$. Note that this linear program makes use of state graphs similar to those in Fig. 3 and Fig. 4. Next, square $[3, 4; 2]$ of size 2 is added. In general, if the procedure divides $[i, i + 1; 2]$ of size 2 and $[i + 1, i + 2; 2]$ of size 2 does not exist, then square $[i + 1, i + 2; 2]$ is added. Thus at this stage there are four squares of size 1 (two of these are, in fact, outside Ω) and one square of size 2. The procedure further divides that square (inside Ω) whose g value is the worst. One continues in this way and takes the worst g-value as an upper bound of the competitive ratio.

An issue regarding the efficiency of the procedure is that the number of states of the state diagram used by the algorithm for a small square (or for the corresponding triangle) becomes large. This leads to potentially excessive computation times for the LP involved. Consider, for example, the triangle $(1, \frac{170}{128}, \frac{213}{128})$ (or the square $[\frac{170}{128}, \frac{213}{128}; \frac{1}{128}]$). It turns out that one needs 514 states for the diagram and substantial computation time in solving the LP is required. However, note that there is a slightly larger triangle, $(1, \frac{4}{3}, \frac{5}{3})$ (or the square $[\frac{4}{3}, \frac{5}{3}; \frac{5}{384}]$), which needs only 12 states to solve the LP (Fig. 7). Thus one can shorten computation time by using $[\frac{4}{3}, \frac{5}{3}; \frac{5}{384}]$ instead of $[\frac{170}{128}, \frac{213}{128}; \frac{1}{128}]$, thereby trading off the g-value of the former (= 1.5606), for that of the latter (= 1.5549). Although we do not have an exact relation between the triangle and the number of states, we have observed that if the ratio of the three sides of the triangle can be represented by three small integers then the number of states is also small. In the procedure, therefore, we do not simply calculate $g(X)$ for a square X, but we attempt to find X' which contains X and has such desirable properties.

Procedure 1 and 2 give a formal description of our method. Each square $X = [a, b; d]$ is represented by $p = (a, b, d, r)$, where r is an upper bound of $g(X)$. The main procedure SQUAREGENERATION divides the square, whose g value is the worst, into four half-sized squares and, if necessary, also creates a new rightmost square of size 2. Then the g-values of those new squares are calculated by procedure

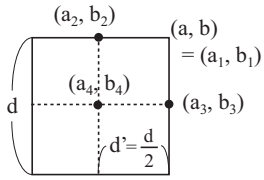

Figure 8. Lines 9-13

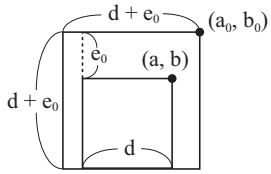

Figure 9. Line 27

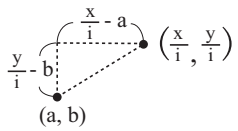

Figure 10. Lines 40-47

CALCULATECR. As discussed above, our method attempts to find a more suitable square, such that the number of states can be kept manageable. More formally, let the current square be $X = [a, b; d]$. Then one seeks to find $\tilde{X} = [\tilde{a}, \tilde{b}; \tilde{d}]$ which contains X and where $\tilde{a}$ can be represented by $\frac{\beta}{\alpha}$, such that both α and β are integers and α is at most 31. (Similarly for $\tilde{b}$.) We have confirmed that the number of states and LP computation time are reasonably small if α is at most this size; for details, see also FINDAPPROXPOINT in Procedure 2. We note that we scan the value of α only in the range from 17 to 31. This is sufficient; for example, $\alpha = 10$ can be covered by $\alpha = 20$. The value $\alpha = 16$ is not needed either since it should have been calculated previously in the subdivision process. If $g(\tilde{X})$ is smaller than the g-value of the original square (of double size), then we use that value as the g-value of X. Otherwise we abandon such an approximation and calculate $g(X)$ directly.

Now suppose that SQUAREGENERATION has terminated. Then for any $p = (a, b, d, r)$ in P, it is guaranteed that $r \leq R_0$. This means that the set of squares which satisfy the conditions (1) to (3) have been created. As mentioned there, we have also created the set of triangles satisfying the conditions of Sec. 2. Thus by Lemma 3, we can conclude:

Theorem 1 *There is an online algorithm for the 2-server 3-point problem whose competitive ratio is at most R_0.*

We now give results of our computer experiments: For the entire area Ω, the current upper bound is 1.5897 (recall that the conjecture is 1.5819). The number N of squares generated is 13285, in which the size m of smallest squares is $1/256$ and the size M of largest squares is 2. We also conducted experiments for small subareas of Ω: (1) For $[5/4, 7/4, 1/16]$: The upper bound is 1.5784 (better than the conjecture but this is not a contradiction since our triangles are restricted). $(N, M, m) = (69, 1/64, 1/128)$. (2) For $[7/4, 9/4, 1/4]$: The upper bound is 1.5825. $(N, M, m) = (555, 1/64, 1/2048)$. (3) For $[10, 11, 1]$: The upper bound is 1.5887. $(N, M, m) = (135, 1/16, 1/32)$. We note that for our computations the LP solver which is part of Mathematica was used for our computations and accuracy to within 5 digits is guaranteed.

5. Concluding Remarks

There are at least two directions for future research: First one might prove that the competitive ratio of the 2-server 3-point problem is analytically at most $e/(e-1) + \varepsilon$. Secondly we wish to extend our current approach (i.e., approximation of infinite point locations by finite ones) to four (or more) points. We have a partial result for the 4-point case where two of the four points are close (obviously it is similar to the 3-point case), but the generalization does not appear easy.

Procedure 1 Procedure SquareGeneration

```
procedure SQUAREGENERATION(R_0)
    p ← (2, 3, 2, C_2 · 2/(2 − 2) = ∞)
    Mark p.
    P ← {p}
    while ∃p = (a, b, d, r) such that r > R_0 do
        p ← the point in P whose r is maximum.
        P ← P\{p}
        Let p = (a, b, d, r)
        d' ← d/2
        a_1 ← a, b_1 ← b
        a_2 ← a − d', b_2 ← b
        a_3 ← a, b_3 ← b − d'
        a_4 ← a − d', b_4 ← b − d'                    ▷ See Fig. 8.
        for i ← 1 to 4 do
            if (a_i, b_i) ∈ Ω then
                r_i ← CALCULATECR(a_i, b_i, d', r)
                P ← P ∪ {(a_i, b_i, d', r_i)}
            end if
        end for
        if p is marked then
            p' ← (a + 1, b + 1, 2, C_{a+1} · a/(a − 2)).
            Mark p'. Unmark p.
            P ← P ∪ {p'}
        end if
    end while
end procedure
procedure CALCULATECR(a, b, d, r)
    (a_0, b_0) ← FINDAPPROXPOINT(a, b)                ▷ See Fig. 9.
    r_0 ← GETCR_FROMLP(a_0, b_0)
    e_0 ← max(a − a_0, b − b_0)
    r̃_0 ← r_0 · a_0/(a_0 − d − e_0)
    if r̃_0 < r_0 then
        return r̃_0
    else
        r_0 ← GETCR_FROMLP(a, b)
        r̃_0 ← r_0 · a_0/(a_0 − d)
        return r̃_0
    end if
end procedure
```

Procedure 2 Procedure FindApproxPoint

```
procedure FINDAPPROXPOINT(a, b)
    e_min ← ∞
    for i ← 31 to 17 do
        x ← ⌈a · i⌉, y ← ⌈y · i⌉, e ← max(x/i − a, y/i − b)
        if e < e_min then
            e_min ← e, i_min ← i, x_min ← x, y_min ← y
        end if
    end for
    return (x_min/i_min, y_min/i_min)
end procedure
```

References

1. Manasse, M.; McGeoch, L. A.; Sleator, D. Competitive algorithms for server problems. *J. Algorithms* **1990**, *11*, 208–230.
2. Koutsoupias, E.; Papadimitriou, C. On the k-server conjecture. *J. ACM* **1995**, *42*, 971–983.
3. Chrobak, M.; Karloff, H.; Payne, T. H.; Vishwanathan, S. New results on server problems. *SIAM J. Discrete Math.* **1991**, *4*, 172–181.
4. Chrobak, M.; Larmore, L. L. An optimal online algorithm for k servers on trees. *SIAM J. Comput.* **1991**, *20*, 144–148.
5. Koutsoupias, E.; Papadimitriou, C. Beyond competitive analysis. In *Proc. 35th FOCS*, pages 394–400. IEEE, **1994**.
6. Bein, W.; Chrobak, M.; Larmore, L. L. The 3-server problem in the plane. In *Proc. 7th European Symp. on Algorithms (ESA)*, volume 1643 of *Lecture Notes in Comput. Sci.*, pages 301–312. Springer, **1999**.
7. Bartal, Y.; Bollobas, B.; Mendel, M. A Ramsey-type theorem for metric spaces and its applications for metrical task systems and related problems. In *Proc. 42nd FOCS*, pages 396–405. IEEE, **2001**.
8. Coppersmith, D.; Doyle, P. G.; Raghavan, P.; Snir, M. Random walks on weighted graphs and applications to on-line algorithms. *J. ACM* **1993**, *40*, 421–453.
9. Karlin, A.; Manasse, M.; McGeoch, L.; Owicki, S. Competitive randomized algorithms for nonuniform problems. *Algorithmica* **1994**, *11*, 542–571.
10. Chrobak, M.; Larmore, L. L.; Lund, C.; Reingold, N. A better lower bound on the competitive ratio of the randomized 2-server problem. *Inform. Process. Lett.* **1997**, *63*, 79–83.
11. Bartal, Y.; Chrobak, M.; Larmore, L. L. A randomized algorithm for two servers on the line. In *Proc. 6th ESA*, Lecture Notes in Comput. Sci., pages 247–258. Springer, **1998**.
12. Bein, W.; Iwama, K.; Kawahara, J.; Larmore, L. L.; Oravec, J. A. A randomized algorithm for two servers in cross polytope spaces. In *Proc. 5th WAOA*, volume 4927 of *Lecture Notes in Computer Science*, pages 246–259. Springer, **2008**.
13. Bein, W.; Larmore, L. L.; Noga, J. Equitable revisited. In *Proc. 15th ESA*, volume 4698 of *Lecture Notes in Computer Science*, pages 419–426. Springer, **2007**.

14. Lund, C.; Reingold, N. Linear programs for randomized on-line algorithms. In *Proc. 5th SODA*, pages 382–391. ACM/SIAM, **1994**.
15. Karlin, A. R.; Kenyon, C.; Randall, D. Dynamic tcp acknowledgement and other stories about $e/(e-1)$. In *Proc. 33rd STOC*, pages 502–509. ACM, **2001**.
16. Reingold, N.; Westbrook, J.; Sleator, D. D. Randomized competitive algorithms for the list update problem. *Algorithmica* **1994**, *11*, 15–32.
17. Fiat, A.; Karp, R.; Luby, M.; McGeoch, L. A.; Sleator, D.; Young, N. E. Competitive paging algorithms. *J. Algorithms* **1991**, *12*, 685–699.
18. Appel, K.; Haken, W. Every planar map is four colorable. *Illinois Journal of Mathematics* **1977**, *21*(5), 429–597.
19. Feige, U.; Goemans, M. X. Approximating the value of two prover proof systems, with applications to max-2sat and max-dicut. In *Proc. 3rd ISTCS*, pages 182–189, **1995**.
20. Goemans, M. X.; Williamson, D. P. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM* **1995**, *42*(6), 1115–1145.
21. Karloff, H.; Zwick, U. A 7/8-approximation algorithm for max 3sat. In *Proc. 38th FOCS*, pages 406–417. IEEE, **1997**.
22. Trevisan, L.; Sorkin, G. B.; Sudan, M.; Williamson, D. P. Gadgets, approximation, and linear programming. *SIAM J. Comput.* **2000**, *29*(6), 2074–2097.
23. Seiden, S. S. On the online bin packing problem. *J. ACM* **2002**, *49*(5), 640–671.
24. Horiyama, T.; Iwama, K.; Kawahara, J. Finite-state online algorithms and their automated competitive analysis. In *Proc. 17th ISAAC*, volume 4288 of *Lecture Notes in Comput. Sci.*, pages 71–80. Springer, **2006**.
25. Chrobak, M.; Larmore, L. L. The server problem and on-line games. In McGeoch, L. A.; Sleator, D. D., editors, *On-line Algorithms*, volume 7 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 11–64. AMS/ACM, **1992**.