Searching in an Unknown Environment: An Optimal Randomized Algorithm for the Cow-Path Problem

By: Ming-Yang Kao, John H. Reif, and Stephen R. Tate

M. Kao, J. H. Reif, and S. R. Tate. "Searching in an Unknown Environment: An Optimal Randomized Algorithm for the Cow-Path Problem", Information and Computation, Vol. 133, No. 1, 1996, pp. 63–80.

Abstract:
Searching for a goal is a central and extensively studied problem in computer science. In classical searching problems, the cost of a search function is simply the number of queries made to an oracle that knows the position of the goal. In many robotics problems, as well as in problems from other areas, we want to charge a cost proportional to the distance between queries (e.g., the time required to travel between two query points). With this cost function in mind, the abstract problem known as the w-lane cow-path problem was designed. There are known optimal deterministic algorithms for the cow-path problem; we give the first randomized algorithm in this paper. We show that our algorithm is optimal for two paths ($w = 2$) and give evidence that it is optimal for larger values of w. Subsequent to the preliminary version of this paper, Kao et al. (in "Proceedings, 5th ACM–SIAM Symposium on Discrete Algorithm," pp. 372-381, 1994) have shown that our algorithm is indeed optimal for all $w \geq 2$. Our randomized algorithm gives expected performance that is almost twice as good as is possible with a deterministic algorithm. For the performance of our algorithm, we also derive the asymptotic growth with respect to w—despite similar complexity results for related problems, it appears that this growth has never been analyzed.

Article:

1. INTRODUCTION

The problem of searching is central to almost all areas of computer science. Variants of searching problems come up often in the study of data structures, database applications, computational geometry, and artificial intelligence. Due to the importance of searching problems, many variants of simple searching have been studied, including searching in unknown environments [2, 7], and searching in the presence of errors [1, 14].

In this paper, we examine the problem of searching in an unknown environment; specifically, we study a problem known as the w-lane cow-path problem. The name comes from the following scenario: Consider a cow, Bessie, standing at a crossroads (referred to as the origin) with w paths leading off into unknown territory. On one of the paths there is a grazing field (the goal) at distance n from the intersection, and all of the other paths go on forever; unfortunately, Bessie's eyesight is not very good—she will not know that she has found the field until she is standing in it (i.e., she cannot see down the road). Clearly Bessie must walk at least distance n to get to the field; if she knows which path to take, she will walk exactly distance n. When Bessie has no prior knowledge of which path the field is on, or of the value n, we would like to know how she can find the field while traveling the least distance possible.

This problem plays an important role in many areas of computer science. The most obvious application is in the area of robotics—when a robot is put in an unknown environment, this exact problem comes up repeatedly. For instance, when a robot is exploring an unknown two dimensional environment (e.g., a mobile robot on the floor of a cluttered warehouse), each time it runs into an obstacle, it should find the closest corner of the obstacle to go around (see, for example, [4]). This robotics problem is just a case of the w-lane cow-path problem with $w = 2$. In addition, this algorithm can be used in the very general context of creating hybrid algorithms. In this case, the different paths represent different base algorithms, and execution alternates between different base algorithms according to search distances given by the cow-path problem. A previous algorithm for this problem was applied in this manner by Fiat et al. in presenting the first competitive algorithm for the online k-server problem [8]. A recent paper by Kao et al. [10] further explores the construction of hybrid algorithms from a set of known algorithms and is based in large part on a preliminary version of this paper [11]. Furthermore, the cow-path problem comes up in artificial intelligence applications where a goal is sought in a largely unknown search space (for an overview of searching in artificial intelligence, see [12]). These examples demonstrate the breadth of applications and fundamental nature of the cow-path problem.

The cow-path problem has much in common with the study of online algorithms, and we use the notion of competitive analysis of online algorithms in order to measure the efficiency of algorithms for the cow-path problem. The competitive ratio for an algorithm solving the cow-path problem is the worst-case ratio of the expected distance traveled by the algorithm to the shortest-path distance from origin to goal. In particular, if the worst-case expected distance traveled by a randomized algorithm is at most $cn + d$, where n is the distance to the goal and d is a fixed constant, then the competitive ratio of this algorithm is c.

In previous work, Baeza-Yates et al. gave an optimal deterministic algorithm for the cow-path problem [2]. As a function of w, the competitive ratio for their algorithm is asymptotically equal to $1 + 2ew$, and for $w = 2$ the ratio is exactly 9. They also prove that their algorithm gives optimal deterministic performance. In other previous work, Chrobak and Larmore have studied a related problem called □Metrical Service Systems," in which requests (which are subsets of points in a metric space) must be served by a single server moving in that metric space [5]. While this problem has some substantial differences from the cow-path problem, there are several similarities, one of which is striking—Chrobak and Larmore's deterministic algorithm for MSS2 (where requests are pairs of points) has competitive ratio 9, just like the deterministic 2-path cow-path problem, and the performance of their randomized algorithm for MSS$_2$ is exactly the same as the performance of the randomized cow-path algorithm that we present in this paper (approximately 4.5911). Chrobak and Larmore leave open the question of whether their algorithms are optimal for MSS$_2$. In one last example of previous work, the cow-path problem has also been studied in the context of game theory by Gal [9]. Gal in fact gives many of the results of the Baeza-Yates et al. paper [2] and of our paper. A main difference between our work and that of Gal is in the focus—our results are self-contained and use results and notation familiar to the theoretical computer science community. In addition, the lower bound proof of this paper presents a new, general purpose lower bound technique that should be useful in proving lower bounds for many other problems. In fact, it was this technique that allowed Kao et al. [10] to extend our optimality proofs to a more powerful result, enhancing both this paper and the work of Gal. Finally, we also analyze the growth rate of the competitive ratio of our algorithm, filling in an important gap in both Gal's work and the original conference version of this paper [11].

In this paper, we give the first randomized algorithm for the cow-path problem, and we give a lower bound proof to show that our algorithm is optimal for $w = 2$. The ratio achieved is a rather complicated value—it is exactly given in terms of the fixed point of a certain equation—and is asymptotically equal to $\kappa w + o(w)$, where κ is a constant value approximately equal to 3.088. For the important case of $w = 2$, the competitive ratio of our algorithm is approximately 4.5911, which is almost twice as good as the best that can be done deterministically. Subsequent to the publication of the conference version of this paper [11], in which we conjectured that our algorithm is also optimal for $w \geq 3$, Kao et al. [10] have given a rather intricate proof showing that our algorithm is in fact optimal for all $w \geq 2$.

A similar problem, known as layered graph traversal, has been studied by Papadimitriou and Yannakakis [13] and Fiat et al. [7]. Layered graph traversal is similar to the cow-path problem, but allows shortcuts between paths without going through the origin, and when one path is explored, information about the other paths may be obtained at no cost. If only deterministic algorithms are considered, then the cow-path problem can be considered a special case of layered graph traversal; however, when randomized algorithms are considered, the problems are fundamentally different. Fiat et al. showed that in layered graph traversal, an exponential (in the number of paths) improvement could be obtained using randomization [7].

2. DEFINITIONS

Let $\mathcal{A}$ be a deterministic algorithm for the cow-path problem. For any goal position g at distance dist(g) from the origin, algorithm $\mathcal{A}$ travels a fixed distance, which we denote cost($\mathcal{A}$, g), to find the goal. We say that algorithm $\mathcal{A}$ has competitive ratio c if, for all goal positions g,

$$\operatorname{cost}(\mathcal{A}, g) \leqslant c \operatorname{dist}(g) + d,$$

where c and d are constants that are independent of the goal position g.

If algorithm $\mathcal{R}$ is a randomized algorithm, then the distance traveled to find a particular goal position is no longer fixed. Instead, cost($\mathcal{R}$, g) is a random variable, and we define the competitive ratio by the expected value of this random variable. In other words, algorithm $\mathcal{R}$ has competitive ratio c if, for all goal positions g,

$$E[\operatorname{cost}(\mathcal{R}, g)] \leqslant c \operatorname{dist}(g) + d, \tag{1}$$

where c and d are constants as before.

In particular, if an algorithm for the cow-path problem has competitive ratio c, then for any goal position that is distance n from the origin, the expected distance that the algorithm has to travel in order to find the goal is at most cn plus some small constant.

3. ALGORITHM

In this section we describe SmartCow, our randomized algorithm for the cow-path problem. SmartCow is a randomized geometric sweep algorithm with geometric ratio $r > 1$, a constant that is fixed for the duration of the algorithm. For ease of reference, assume that the w paths are labeled with integers 0, 1, ..., w & 1. The general outline of SmartCow can be found in Fig. 1; the analysis of the competitive ratio will be done in terms of the constant r, and in Section 5 we will see how to find the best possible r.

```
$\sigma \leftarrow$ A random permutation of $\{0, 1, 2, \cdots, w-1\}$;
$\epsilon \leftarrow$ A random real uniformly chosen from $[0, 1)$;
$d \leftarrow r^{\epsilon}$;
$p \leftarrow 0$;
repeat
        Explore path $\sigma(p)$ up to distance $d$;
        if goal not found then return to origin;
        $d \leftarrow d \cdot r$;
        $p \leftarrow (p+1) \bmod w$;
until goal found;
```

FIG. 1. Algorithm SmartCow, for parameter $r > 1$.

It should be noted that the use of randomization is very limited; randomization is used only at the very beginning of the search, in order to pick a random permutation and a random initial search distance." The algorithm never needs access to a random number generator once the search has begun. We define the function

$$R(r, w) = 1 + \frac{2}{w} \cdot \frac{1 + r + r^2 + \cdots + r^{w-1}}{\ln r}, \tag{2}$$

which we will next prove to be the competitive ratio of algorithm SmartCow.

Theorem 3.1. *For any fixed $r > 1$, Algorithm SmartCow has competitive ratio $R(r, w)$.*

Proof. For a given goal position, let n denote the distance from the origin to the goal, and let q be the path on which the goal lies. If $n < 1$ we can handle this as a special case in the analysis—the distance traveled is clearly at most $(r^w - 1)1(r - 1)$, which is independent of n, and so can be entirely covered by the constant term d in Eq. (1). Therefore, for the remainder of this proof we assume that $n \geq 1$. Furthermore, let k be an integer, and let δ be a real value satisfying $0 \leq \delta < 1$, where k and δ are such that $n = r^{k+\delta}$.

Notice from Fig. 1 that SmartCow proceeds in stages, where at stage $i \in \{0, 1, 2, \ldots\}$ the algorithm sweeps distance $r^{i+\varepsilon}$ on path $\sigma(i \bmod w)$. Let m be the first stage where SmartCow sweeps distance at least r^k on the same path as the goal. More formally, m is the least integer such that $m \geq k$ and $\sigma(m \bmod w) = q$. The value m always satisfies $k \leq m \leq k + w - 1$.

Case 1. $m \geq k + 1$. In this case, the sweep distance is at least r^{k+1} at stage m, so SmartCow always finds the goal on stage m. If D is the random variable denoting the distance traveled by our algorithm, then it is easy to see that when $m = c \geq k + 1$

$$D = 2\sum_{i=0}^{c-1} r^{i+\varepsilon} + n = \frac{2r^{\varepsilon}(r^c - 1)}{r-1} + n,$$

and the expected value is easily calculated as

$$E[D \mid m = c] = \frac{2(r^c - 1)}{r-1}\, E[r^{\varepsilon} \mid m = c] + n.$$

Calculating $E[r^{\varepsilon} \mid m = c]$ is relatively straightforward. The density function for r^{ε} is calculated from the fact that ε is uniformly distributed, giving

$$E[r^{\varepsilon} \mid m = c] = E[r^{\varepsilon}] = \int_1^r x \cdot \frac{1}{x \ln r}\, dx = \frac{r-1}{\ln r}.$$

Thus, the resulting expected distance traveled in this case is

$$E[D \mid m = c] = \frac{2(r^c - 1)}{\ln r} + n.$$

Case 2. $m = k$. In this case, SmartCow may or may not find the goal on sweep m, depending on whether or not $\varepsilon \geq \delta$. Let F denote the event that SmartCow finds the goal at stage m. Then

$$\begin{aligned} E[D \mid m = k] &= \text{Prob}(F) E\left[2\sum_{i=0}^{k-1} r^{i+\varepsilon} + n \,\middle|\, F \right] \\ &\quad + \text{Prob}(\bar{F}) E\left[2\sum_{i=0}^{k+w-1} r^{i+\varepsilon} + n \,\middle|\, \bar{F} \right] \\ &= \text{Prob}(F)\, \frac{2(r^k - 1)}{r-1}\, E[r^{\varepsilon} \mid F] + \text{Prob}(\bar{F})\, \frac{2(r^{k+w} - 1)}{r-1}\, E[r^{\varepsilon} \mid \bar{F}] + n \\ &= \frac{2}{r-1}\, [\text{Prob}(F)(r^k - 1)\, E[r^{\varepsilon} \mid F] \\ &\quad + \text{Prob}(\bar{F})(r^{k+w} - 1)\, E[r^{\varepsilon} \mid \bar{F}]] + n. \end{aligned}$$

In this case, $E[r^{\varepsilon} \mid F]$ and $E[r^{\varepsilon} \mid \bar{F}]$ can be found as

$$E[r^{\varepsilon} \mid F] = \int_{r^{\delta}}^{r} x \cdot \frac{1}{\text{Prob}(F) x \ln r}\, dx = \frac{r - r^{\delta}}{\text{Prob}(F) \ln r};$$

$$E[r^{\varepsilon} \mid \bar{F}] = \int_{1}^{r^{\delta}} x \cdot \frac{1}{\text{Prob}(\bar{F}) x \ln r}\, dx = \frac{r^{\delta} - 1}{\text{Prob}(\bar{F}) \ln r}.$$

Using these values,

$$E[D|m = k] = \frac{2}{(r-1)\ln r}\, [(r - r^{\delta})(r^k - 1) + (r^{\delta} - 1)(r^{k+w} - 1)] + n.$$

The competitive ratio of SmartCow depends on the overall, or unconditional, expected distance $E[D]$. This is calculated by combining the above results, using the formula

$$E[D] = \sum_{i=k}^{k+w-1} \text{Prob}(m = i)\, E[D \mid m = i].$$

At the beginning of the search, the algorithm chooses a random permutation σ, so Prob($m = i$) = $1/w$ for every i such that $k \leq i \leq k + w - 1$.

Therefore, the above equation can be expanded to

$$E[D] = \frac{2}{w(r-1)\ln r}\left[(r-r^{\delta})(r^{k}-1)+(r^{\delta}-1)(r^{k+w}-1)\right]$$

$$+\frac{n}{w}+\sum_{i=k+1}^{k+w-1}\left[\frac{2(r^{m}-1)}{w\ln r}+\frac{n}{w}\right]$$

$$=\frac{2}{w(r-1)\ln r}\left[(r-r^{\delta})(r^{k}-1)+(r^{\delta}-1)(r^{k+w}-1)\right.$$

$$\left.+(r-1)\sum_{i=k+1}^{k+w-1}(r^{m}-1)\right]+n$$

$$=\frac{2}{w(r-1)\ln r}\left[r^{k+\delta}(r^{w}-1)-w(r-1)\right]+n$$

$$\leqslant\left[\frac{2(r^{w}-1)}{w(r-1)\ln r}+1\right]n.$$

The competitive ratio is simply the expected distance traveled ($E[D]$) divided by n:

$$1+\frac{2(r^{w}-1)}{w(r-1)\ln r}=1+\frac{2}{w}\cdot\frac{1+r+r^{2}+\cdots+r^{w-1}}{\ln r}=R(r,w),$$

which is exactly the value claimed in the theorem.

From the preceding theorem, it is difficult to see how the performance of algorithm SmartCow compares to that of the optimal deterministic algorithm given by Baeza-Yates et al. [2]. For example, when $w = 2$ their algorithm has a competitive ratio of 9, while Theorem 3.1 states that SmartCow has competitive ratio $1 + (1 + r)/\ln r$. In Section 5 we will see how to choose r so that the competitive ratio of SmartCow is approximately 4.5911, or almost twice as good as the deteministic algorithm. In the next section we will see that this is in fact the best ratio that can be achieved by *any* randomized algorithm.

4. LOWER BOUND

To prove our lower bound for randomized algorithms, we appeal to Yao's corollary to the famous von Neumann minimax principle [16]. In particular, we define a probability distribution for inputs to the cow-path problem and then lower bound the performance of any *deterministic* algorithm on this input distribution. Yao's result states that this lower bound must also be a lower bound for the expected performance of any randomized algorithm on its worst-case input.

We actually use a family of probability distribution functions, parameterized by $\varepsilon > 0$. We will denote a particular distribution function by $f_{\varepsilon,w}$, and we will use Opt(ε, w) to denote the optimal competitive ratio of any deterministic algorithm with input distribution $f_{\varepsilon,w}$. Our goal will be to show that $lim_{\varepsilon\to 0}$ Opt(ε, w) exists, and give a value for this limit. The following lemma shows that this limit is a lower bound for the original problem.

Lemma 4.1. *Let* OptR(w) *denote the optimal competitive ratio for any randomized cow-path algorithm on inputs with w paths. If* $\ell = lim_{\varepsilon\to 0}$ Opt(ε, w), *then* OptR(w) $\geq \ell$.

Proof. For the sake of contradiction, assume that there is a randomized cow-path algorithm that achieves competitive ratio $p < \ell$. Let $\delta = (\ell - p)/2$. Now by the formal definition of the limit, there exists an ε_0 such that for all $\varepsilon < \varepsilon_0$, $|$ Opt(ε, w) $- \ell | \leq \delta$. In other words, for any $\varepsilon < \varepsilon_0$,

$$\mathrm{Opt}(\varepsilon,w)\geqslant \ell-\delta=\frac{2\ell-(\ell-\rho)}{2}=\frac{\ell+\rho}{2}>\rho.$$

But, by Yao's lemma, this implies that OptR(w) > p, which contradicts to the original assumption that there exists a randomized algorithm with competitive ratio p.

Now we define the density function $f_{\varepsilon,w}$. To specify the position of the goal, we need to specify both the path on which the goal lies and the distance down that path to the goal. For all values of =, the path is chosen uniformly from all possible paths. Thus, we will use $f_{\varepsilon,w}$ to denote only the distance down the chosen path to the goal. The density function we use is

$$f_{\varepsilon,w}(x) = \begin{cases} \varepsilon x^{-(1+\varepsilon)} & \text{if} \quad x \geqslant 1; \\ 0 & \text{otherwise.} \end{cases}$$

Any deterministic algorithm can be defined by a sequence (s_0, p_0), (s_1, p_1), ..., (s_k , p_k), ... , where s_k is the distance of the kth sweep and p_k is the path on which the kth sweep is taken. In fact, since the goal is placed on a uniformly chosen ray, we can assume that the sequence of path explorations goes in a fixed cyclic order. Without loss of generality, we assume that p_k = (k mod w), and then the algorithm is completely specified by the sequence s0, s_1, ... , s_k, ⋯. Since the distance from the origin to the goal is at least one, we can safely assume that $s_0 \geq 1$. In fact, by adding an extra search probe in the beginning, we can assume that $s_0 = 1$; the cost of this extra probe is just an additive constant, which does not affect the competitive ratio. Using this notation, we can prove the following lemma.

Lemma 4.2. *Let algorithm* A *be a deterministic algorithm defined by the sequence* $s_0, s_1, \ldots, s_k, \ldots$ *. For input distribution* $f_{\varepsilon,w}$ *, the expected competitive ratio of* A *is*

$$1 + \frac{2\varepsilon}{w(1+\varepsilon)} \left(\sum_{i=0}^{w-2} (w-1-i)s_i + \sum_{i=0}^{\infty} s_i^{-(1+\varepsilon)} \sum_{j=0}^{w-1} s_{i+j} \right).$$

Proof. The position of the goal can be specified by defining two random variables. The first, P, is uniformly distributed over {0, 1, ..., w - 1 }and determines the path that the goal lies on. The second random variable, D, is distributed according to $f_{\varepsilon,w}$, defined above, and represents the distance from the origin to the goal.

We will also define some conditions C_i for i = 0, 1, 2, ..., where C_i is true exactly when algorithm A finds the goal on sweep i. More formally,

$$C_i \text{ is true if and only if } \begin{cases} 1 \leqslant D \leqslant s_i \text{ and } P = i & \text{when} \quad 0 \leqslant i < w; \\ s_{i-w} < D \leqslant s_i \text{ and } P \equiv i (\text{mod } w) & \text{when} \quad i \geqslant w. \end{cases}$$

Notice that the conditions C_i partition the space of all possible goal positions, so if we let p_i = Prob(C_i), then it should be clear that $\Sigma_{i=0}^{\infty} p_i = 1$. Furthermore, if R is a random variable denoting the competitive ratio achieved by algorithm A, then

$$E[R] = \sum_{i=0}^{\infty} p_i E[R \mid C_i]. \tag{3}$$

In computing the expected values $E[\ R \mid C_i]$ there are three cases: $i = 0$, $1 \leq i < w$, and $i \geq w$. We present the analysis for $i \geq w$ below; the remaining cases are similar.

Computing $E[R \mid C_i]$, we know that C_i holds, so the distance traveled by the algorithm is

$$\sum_{j=0}^{i-1} (2s_j) + D.$$

Dividing by D, we see that the expected competitive ratio under this condition is given by

$$E[R \mid C_i] = E\left[\sum_{j=0}^{i-1} \frac{2s_j}{D} + 1 \mid C_i \right] = E\left[\frac{1}{D} \middle| C_i \right] \sum_{j=0}^{i-1} (2s_j) + 1. \tag{4}$$

To calculate $E[(1/D) \mid C_i]$, we simply refer back to the distribution for D, scale this by p_i since we want the conditional expectation, and integrate to find the expected value. In other words,

$$E\left[\frac{1}{D}\,\middle|\, C_i\right]=\int_{s_{i-w}}^{s_i}\frac{1}{x}\,\frac{\varepsilon}{wp_i}\,x^{-(1+\varepsilon)}\,dx=\frac{\varepsilon}{wp_i(1+\varepsilon)}\,(s_{i-w}^{-(1+\varepsilon)}-s_i^{-(1+\varepsilon)}).$$

Combining this with Eq. (4) gives the conditional expected competitive ratio.

Summarizing all cases for the conditional expectation,

$$E[R\mid C_i]=\begin{cases}1 & \text{if}\quad i=0;\\ \dfrac{2\varepsilon}{wp_i(1+\varepsilon)}\left(1-s_i^{-(1+\varepsilon)}\right)\sum\limits_{j=0}^{i-1}s_j+1 & \text{if}\quad 1\leqslant i<w;\\ \dfrac{2\varepsilon}{wp_i(1+\varepsilon)}\,(s_{i-w}^{-(1+\varepsilon)}-s_i^{-(1+\varepsilon)})\sum\limits_{j=0}^{i-1}s_j+1 & \text{if}\quad i\geqslant w.\end{cases}$$

Combining these results with Eq. (3) gives (after some algebraic manipulation)

$$\frac{2\varepsilon}{w(1+\varepsilon)}\left(\sum_{i=0}^{w-2}(w-1-i)s_i+\sum_{i=0}^{\infty}s_i^{-(1+\varepsilon)}\sum_{j=0}^{w-1}s_{i+j}\right),$$

which is exactly what we are proving.

Using the two preceding lemmas, we can prove that the algorithm of the previous section is optimal for $w = 2$.

Theorem 4. 1. *For $w = 2$, the optimal competitive ratio is given by*

$$\min_{r>1}\left\{1+\frac{1+r}{\ln r}\right\}.$$

Since this ratio is achievable by the algorithm of the previous section, the algorithm SmartCow *is optimal.*

Proof. Assume that the values s_0, s_1, ..., s_k, ..., define the optimal deterministic algorithm for a fixed ε, and let Opt(ε, 2) denote the competitive ratio given in Lemma 4.2. Rewrite this formula in cleaner form for $w = 2$:

$$\text{Opt}(\varepsilon,\,2)=1+\frac{\varepsilon}{1+\varepsilon}\left(s_0+\sum_{i=0}^{\infty}\frac{s_i+s_{i+1}}{s_i^{1+\varepsilon}}\right).$$

For a fixed ε, to lower bound this equation, we need only find a lower bound for

$$R(\varepsilon)=\sum_{i=0}^{\infty}\frac{s_i+s_{i+1}}{s_i^{1+\varepsilon}}=\frac{s_0+s_1}{s_0^{1+\varepsilon}}+\sum_{i=1}^{\infty}\frac{s_i+s_{i+1}}{s_i^{1+\varepsilon}}=1+s_1+\sum_{i=1}^{\infty}\frac{s_i+s_{i+1}}{s_i^{1+\varepsilon}}$$

(recall that $s_0 = 1$ is fixed).

By setting $t_i = (s_{i+1})/s_1$, we obtain a new sequence with $t_0 = 1$. The above sum can be written in terms of this new sequence as

$$1+s_1+s_1^{-\varepsilon}\sum_{i=0}^{\infty}\frac{t_i+t_{i+1}}{t_i^{1+\varepsilon}}.$$

But this is easily lower bounded by

$$R(\varepsilon)=1+s_1+s_1^{-\varepsilon}\sum_{i=0}^{\infty}\frac{t_i+t_{i+1}}{t_i^{1+\varepsilon}}\geqslant 1+s_1+s_1^{-\varepsilon}R(\varepsilon).$$

In other words,

$$R(\varepsilon)\geqslant\frac{1+s_1}{1-s_1^{-\varepsilon}},$$

so

$$\text{Opt}(\varepsilon,\,2)\geqslant 1+\frac{\varepsilon}{1+\varepsilon}\left(1+\frac{1+s_1}{1-s_1^{-\varepsilon}}\right).$$

By setting $s_i = s_1^i$ and recalling Lemma 4.2, we see that the geometric sweep algorithm has exactly the competitive ratio stated above as a lower bound. In other words, the above is not just a lower bound, it is in fact the exact optimal value when minimized over s_1. So for fixed ε,

$$\text{Opt}(\varepsilon,\,2)=\min_{r>1}\left\{1+\frac{\varepsilon}{1+\varepsilon}\left(1+\frac{1+r}{1-r^{-\varepsilon}}\right)\right\}.$$

By Lemma 4. 1, we know that OptR(2) ≥ $lim_{\varepsilon\to 0}$ Opt(ε, 2), so we can bound OptR(2) by

$$\begin{aligned}\mathrm{OptR}(2) &\geqslant \lim_{\varepsilon\to 0}\ \min_{r>1}\left\{1+\frac{\varepsilon}{1+\varepsilon}\left(1+\frac{1+r}{1-r^{-\varepsilon}}\right)\right\}\\ &=\min_{r>1}\ \lim_{\varepsilon\to 0}\left\{1+\frac{\varepsilon}{1+\varepsilon}\left(1+\frac{1+r}{1-r^{-\varepsilon}}\right)\right\}\\ &=\min_{r>1}\left\{1+\frac{1+r}{\ln r}\right\}.\end{aligned}$$

The last line above is exactly the bound claimed in the theorem statement.

Subsequent to the conference publication of this paper, Kao et al. used similar reasoning to show that, in fact, SmartCow is optimal for all $w \geq 2$ [10]. Their proof is closely related to the one just presented, using small variants of Lemmas 4.1 and 4.2—an important contribution of their paper is a very intricate and involved proof that replaces our Theorem 4.1 and works for all $w \geq 2$.

5. MINIMIZING THE COMPETITIVE RATIO

Recall from Theorem 3.1 that the algorithm SmartCow has competitive ratio

$$R(r, w) = 1 + \frac{2}{w}\cdot\frac{1+r+r^2+\cdots+r^{w-1}}{\ln r},$$

where r is a fixed algorithm parameter. In other words, for a fixed w SmartCow is really a *class* of algorithms, indexed by the parameter r. In order to get the best performance possible, we would like to pick a value of r that minimizes $R(r, w)$.

Theorem 5.1. *The unique solution of the equation*

$$\ln r = \frac{1+r+r^2+\cdots+r^{w-1}}{r+2r^2+3r^3+\cdots+(w-1)r^{w-1}} \qquad (5)$$

for r > 1, denoted by r_w^*, gives the minimum value for R(r, w).

Proof. To minimize $R(r, w)$ for a fixed w, we need only minimize the part that depends on r. Call this function $f_w(r)$, where

$$f_w(r) = \frac{1+r+r^2+\cdots+r^{w-1}}{\ln r}.$$

This function is continuous, and $f_w(r)$ goes to positive infinity when either end of the interval $(1, \infty)$ is approached. Therefore, any minimum of the function on this interval must be a local minimum, and we can find this by taking a derivative:

$$f'_w(r) = \frac{(r+2r^2+\cdots+(w-1)r^{w-1})\ln r - (1+r+\cdots+r^{w-1})}{(\ln r)^2}.$$

The denominator is non-zero and finite for all r ∈ (1, ∞), and the numerator is zero exactly when Eq. (5) is true. In other words, the minimizing r must satisfy Eq. (5)—by showing that there is only one such r, we will have proved the theorem.

We need to show that Eq. (5) has exactly one solution for $r > 1$. To see this, first note that the function ln r is monotonically increasing for $r > 1$. Next, we will show that the right-hand side of Eq. (5) is monotonically *decreasing*, so it follows that Eq. (5) can have *at most* one solution. To see this, consider the right-hand side of Eq. (5):

$$g_w(r) = \frac{1+r+r^2+\cdots+r^{w-1}}{r+2r^2+3r^3+\cdots+(w-1)r^{w-1}}. \qquad (6)$$

Taking the derivative with respect to r gives

$$g'_w(r) = \frac{\left[\begin{array}{l} r(1+2r+\cdots+(w-1)r^{w-2})^2 \\ \qquad -(1+r+\cdots+r^{w-1})(1+4r+\cdots+(w-1)^2 r^{w-2}) \end{array}\right]}{(r+2r^2+\cdots+(w-1)r^{w-1})^2}.$$

The denominator of $g'_w(r)$ is clearly positive and non-zero for $r > 1$, and the numerator can be written as a polynomial in r. After some algebraic manipulation, it is discovered that the numerator of $g'_w(r)$ can be written as $\sum_{i=0}^{2w-2} c_k r^k$, where the coefficients c_k are

$$c_k = \begin{cases} -\dfrac{(k+1)(k+2)(k+3)}{6} & \text{for} \quad 0 \leqslant k \leqslant w-2; \\ -\dfrac{(2w-k-3)(2w-k-2)(2w-k-1)}{6} & \text{for} \quad w-1 \leqslant k \leqslant 2w-2. \end{cases}$$

TABLE 1

Approximate Values for Small *w*

w	r_w^*	Competitive ratio of SmartCow	Optimal deterministic ratio
2	3.59112	4.59112	9.00000
3	2.01092	7.73232	14.5
4	1.62193	10.84181	19.96296
5	1.44827	13.94159	25.41406
6	1.35020	17.03709	30.85984
7	1.28726	20.13033	36.30277

Clearly, all these coefficients are negative, so for r > 1 the numerator of $g'_w(r)$, and hence $g'_w(r)$ itself, is negative. In other words, we have shown that $g_w(r)$ is monotonically decreasing for $r > 1$.

Now that we have shown that Eq. (5) has at most one solution, we will show that it has *at least* one solution. To see this, consider the function

$$\ln r - \frac{1+r+r^2+\cdots+r^{w-1}}{r+2r^2+3r^3+\cdots+(w-1)r^{w-1}}.$$

For any fixed w, this function is clearly negative for $r = 1$ and positive in the limit as $r \to \infty$. Furthermore, since the function is continuous, it must have a root in the interval $(1, \infty)$. Thus we have proved that Eq. (5) has exactly one solution for r > 1.

The value r_w^* can be found for any given w from Eq. (5) using standard numerical techniques, and using this value we can construct the best algorithm from the family of algorithms described by SmartCow. Approximate values for small values of w are given in Table 1, with the optimal deterministic ratio shown for reference.

Due to the results of Section 4 and of Kao et al. [10], the competitive ratios shown in Table 1 are in fact optimal for randomized algorithms.

6. GROWTH WITH THE NUMBER OF PATHS

In this section, we consider the growth of the competitive ratio of algorithm SmartCow as the number of paths grows. Recall that $R(r, w)$ was defined in Eq. (2) and shown to be the competitive ratio of algorithm SmartCow, and that in Theorem 5.1 we showed that for each w there is a unique $r > 1$ (called r_w^*) that gives the best performance for algorithm SmartCow. We will use $OR(w) = R(r_w^*, w)$ to denote the optimum performance of SmartCow for w values. Furthermore, we define a special constant κ to be the value

$$\kappa = \min_{d>0} \left[2\,\frac{e^d - 1}{d^2} \right] \approx 3.088.$$

We will show that the competitive ratio of algorithm SmartCow is $\kappa w + o(w)$.

In our proof, we will make use of the following easily verified inequalities. For all $x > 0$,

$$e^{1-1/2x} < \left(1+\frac{1}{x}\right)^x < e \qquad (7)$$

and

$$\ln(1+x) \geqslant x - \frac{x^2}{2}. \qquad (8)$$

We are now prepared to prove an upper bound on the competitive ratio of algorithm SmartCow.

LEMMA 6. 1. $OR(w) \leq \kappa w + \Theta(1)$.

Proof. Fix some constant $c > 0$ and define the sequence of values $r_w = 1 + (c/w)$. From inequality (7) we know that

$$e^{(1-(c/(2w)))c} < (r_w)^w < e^c.$$

Now clearly $OR(w) \leq R(r_w, w)$, so for $w > c/2$ we can derive

$$\begin{aligned} OR(w) &\leqslant 1 + 2\,\frac{e^c - 1}{(c^2/w)(1-(c/(2w)))} \\ &= 1 + 2\,\frac{e^c-1}{c^2}\,w\left(1+\frac{c}{2w-c}\right) \\ &= 2\,\frac{e^c-1}{c^2}\,w + \Theta(1). \end{aligned}$$

This is true for any arbitrarily chosen c, so in particular this is true for the c that minimizes the constant of the linear term. Therefore,

$$OR(w) \leqslant \min_{c>0}\left[2\,\frac{e^c-1}{c^2}\right] w + \Theta(1) = \kappa w + \Theta(1),$$

as claimed in the lemma statement.

Lemma 6.1 gives an upper bound on the growth of the competitive ratio with the number of paths, and we now show a similar lower bound; however, first we need a preliminary result bounding r_w^*.

LEMMA 6.2 For any $w \geq 5$, $r_w^* \leq 1 + (5/w)$.

Proof. From Theorem 5.1 we know that for any w the optimizing r_w^* satisfies Eq. (5) and that the right-hand side of this equation, named $g_w(r)$ in Eq. (6), is monotonically decreasing, while the left-hand side is monotonically increasing. Therefore, if we can show that for all $w \geq 5$,

$$\ln\left(1+\frac{5}{w}\right) \geqslant g_w\left(1+\frac{5}{w}\right),$$

then we know that for all $w \geq 5$, $r_w^* \leq 1 + (5/w)$.

To prove this, first notice that since $g_w(r)$ is monotonically decreasing for all $r \geq 1$, we can bound

$$g_w\left(1+\frac{5}{w}\right) \leqslant g_w(1) = \frac{2}{w-1}.$$

Next, from inequality (8) we can derive, for $w \geq 5$,

$$\begin{aligned} \ln\left(1+\frac{5}{w}\right) &\geqslant \frac{5}{w}\left(1-\frac{5}{2w}\right) = \frac{5}{w-1}\left(1-\frac{1}{w}\right)\left(1-\frac{5}{2w}\right) \\ &\geqslant \frac{5}{w-1}\cdot\frac{4}{5}\cdot\frac{1}{2} = \frac{2}{w-1}. \end{aligned}$$

Combining these two bounds, we get

$$\ln\left(1+\frac{5}{w}\right)\geqslant\frac{2}{w-1}\geqslant g_w\left(1+\frac{5}{w}\right),$$

so from the discussion at the beginning of the proof it follows that

$$r_w^*\leqslant 1+\frac{5}{w}$$

as claimed.

LEMMA 6.3. $OR(w) \geq \kappa w - o(w)$.

Proof. First, define $\varepsilon_w = r_w^* - 1$ (so $r_w^* = 1 + \varepsilon_w$). Now notice that $(r_w^*) = (1 + \varepsilon_w)$w can be bounded using inequality (7) as

$$e^{(1-\varepsilon_w/2)\varepsilon_w w}<(r_w^*)^w<e^{\varepsilon_w w}.$$

Thus we can bound

$$\begin{aligned}OR(w)=R(r_w^*,\ w)=R(1+\varepsilon_w,\ w)&\geqslant 1+2\,\frac{e^{(1-\varepsilon_w/2)\varepsilon_w w}-1}{\varepsilon_w^2 w}\\&=1+2\,\frac{e^{(1-\varepsilon_w/2)\varepsilon_w w}-1}{(\varepsilon_w w)^2}\,w.\end{aligned}$$

Now, let $d_w = (1 - \varepsilon_w/2)\varepsilon_w w$, so that the above becomes

$$OR(w)\geqslant 1+2\,\frac{e^{d_w}-1}{d_w^2}\,w\left(1-\frac{\varepsilon_w}{2}\right)^2>\min_{d>0}\left[2\,\frac{e^d-1}{d^2}\right]\,w\left(1-\frac{\varepsilon_w}{2}\right)^2=\kappa w\left(1-\frac{\varepsilon_w}{2}\right)^2.$$

Next, let δ be an arbitrary positive constant. From Lemma 6.2 we know that $\varepsilon_w \leq 5/w$ for $w \geq 5$, so for all $w \geq \max(3/\delta, 5)$ we have

$$\left(1-\frac{\varepsilon_w}{2}\right)^2\geqslant\left(1-\frac{5}{2w}\right)^2\geqslant 1-\frac{3}{w}\geqslant 1-\delta,$$

which implies that

$$OR(w)\geqslant\kappa w\left(1-\frac{\varepsilon_w}{2}\right)^2\geqslant\kappa w(1-\delta).$$

Since this is true for arbitrarily small δ, this then implies that $OR(w) \geq \kappa w$ & $o(w)$, as claimed in the lemma.

The main theorem of this section is a direct and obvious consequence of Lemmas 6.1 and 6.3. Of the several definitions of asymptotic notation, we use the standard definition in which $f(n)$ is $o(g(n))$ if for any constant $c > 0$ there is a constant $n_0 \geq 1$ such that $|f(n)| < cg(n)$ for all $n \geq n_0$—the absolute value on $f(n)$ is necessary for the following theorem to be an exact statement.

Theorem 6.1. *The competitive ratio for algorithm* SmartCow *is* $\kappa w+o(w)$, *where*

$$\kappa=\min_{d>0}\left[2\,\frac{e^d-1}{d^2}\right]\approx 3.088.$$

The use of $o(w)$ in the above theorem is due entirely to the lower bound on the growth rate. As far as the algorithm's performance goes, Lemma 6.1 shows that it is perfectly valid (and somewhat stronger than Theorem 6.1) to say that the competitive ratio is *at most* $\kappa w + \Theta(1)$. For the sake of comparison, recall that the optimal deterministic algorithm of Baeza-Yates et al. [2] has competitive ratio $2ew + \Theta(1)$, or approximately $5.437w + \Theta(1)$.

7. CONCLUSIONS

In this paper we have given a new randomized algorithm, SmartCow, for the cow-path problem. We analyzed the competitive ratio of SmartCow and showed that randomization gives our algorithm almost a factor of 2 improvement over the best possible deterministic algorithm. Furthermore, we have shown that for the important two-path problem our algorithm is an optimal randomized algorithm. The lower bound proof of Section 4 includes a general form for lower bounds when $w \geq 2$, but a closed form was obtained only for $w = 2$ (showing that SmartCow is optimal for $w = 2$). This was recently extended by Kao et al. [10], who gave an involved proof showing that SmartCow is in fact optimal for all $w \geq 2$.

REFERENCES

1. Aslam, J. A., and Dhagat, A. (1991), Searching in the presence of linearly bounded errors, in “Proceedings, 23rd ACM Symposium on Theory of Computing," pp. 486-493.
2. Baeza-Yates, R. A., Culberson, J. C., and Rawlins, G. J. E. (1993), Searching in the plane, Inform. and Comput. 16, 234-252.
3. Bentley, J. L., and Yao, A. C.-C. (1976), An almost optimal algorithm for unbounded searching, Inform. Process. Lett. 5, 82-87.
4. Blum, A., Raghavan, P., and Schieber, B. (1991), Navigating in unfamiliar geometric terrain, in “Proceedings, 23rd ACM Symposium on Theory of Computing," pp. 494-504.
5. Chrobak, M., and Larmore, L. (1991), The server problem and on-line games, in “On-Line Algorithms: Proceedings of a DIMACS Workshop," pp. 11-64, American Math. Society.
6. Deng, X., and Papadimitriou, C. H. (1990), Exploring an unknown graph, in “Proceedings, 31st IEEE Symposium on Foundations of Computer Science," pp. 355-361.
7. Fiat, A., Foster, D. P., Karloff, H., Rabani, Y., Ravid, Y., and Vishwanathan, S. (1991), Competitive algorithms for layered graph traversal, in “Proceedings, 32nd IEEE Symposium on Foundations of Computer Science," pp. 288-297.
8. Fiat, A., Rabani, Y., and Ravid, Y. (1990), Competitive k-server algorithms, in “Proceedings, 31st IEEE Symposium on Foundations of Computer Science," pp. 454-463.
9. Gal, S. (1980), “Search Games," Academic Press, New York.
10. Kao, M. Y., Ma, Y., Sipser, M., and Yin, Y. (1994), Optimal constructions of hybrid algorithms, in “Proceedings, 5th ACM-SIAM Symposium on Discrete Algorithms," pp. 372-381.
11. Kao, M. Y., Reif, J. H., and Tate, S. R. (1993), Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem, in “Proceedings, 4th ACM-SIAM Symposium on Discrete Algorithms," pp. 441-447.
12. Pearl, J. (1984), “Heuristics: Intelligent Search Strategies for Computer Problem Solving," Addison-Wesley, Reading, MA.
13. Papadimitriou, C. H., and Yannakakis, M. (1989), Shortest paths without a map, Theoret. Comput. Sci. 84, 127-150.
14. Rivest, R. L., Meyer, A. R., Kleitman, D. J., Winklmann, K., and Spencer, J. (1980), Coping with errors in binary search procedures, J. Comput. System Sci. 20, 396-404.
15. Sleator, D. D., and Tarjan, R. E. (1985), Amortized efficiency of list update and paging rules, Comm. ACM 28, 202-208.
16. Yao, A. (1977), Probabilistic computations: Towards a unified measure of complexity, in “Proceedings, 18th IEEE Symposium on Foundations of Computer Science," pp. 222-227.