Chapter 47
Searching in an Unknown Environment: An Optimal Randomized Algorithm for the Cow-Path Problem

Ming-Yang Kao*† John H. Reif*‡ Stephen R. Tate*‡

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, and we give the first randomized algorithms in this paper. We show that our algorithm is optimal for two paths ($w = 2$), and give evidence that it is indeed optimal for larger values of w. The randomized algorithms give expected performance that is almost twice as good as is possible with a deterministic algorithm.

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 [2,6], and searching in the presence of errors [1,10].

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 won't know that she has found the field until she is standing in it (i.e., she can't 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, 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 exploring in an unknown two dimensional environment (e.g., a mobile robot on the floor of a cluttered warehouse), each time the robot 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 to this important application, a previous algorithm for this problem was used by Fiat, Rabani, and Ravid in presenting the first competitive algorithm for the online k-server problem [7]. Furthermore, this 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 [8]).

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 $cn + d$, where n is the distance to the goal and d is a fixed constant, then the

*Department of Computer Science, Duke University, Durham, NC 27706

†Research supported in part by NSF Grant CCR-9101385.

‡Research supported in part by DARPA/ISTO Contracts N00014-88-K-0458 and N00014-91-J-1985, NASA subcontract 550-63 of prime contract NAS5-30428, and US-Israel Binational NSF Grant 88-00282/2, and US-Israel Binational NSF Grant 88-00282/2

competitive ratio of this algorithm is c.

In previous work, Baeza-Yates, Culberson, and Rawlins gave an optimal deterministic algorithm for this 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. While their algorithm is optimal for deterministic algorithms, we can do better by using randomization. In this paper, we give the first randomized algorithms for the cow-path problem, and give lower bounds to show that for $w = 2$ our algorithm is optimal. The ratio achieved is a rather complicated value — it is given in terms of the fixed point of a certain equation. For $w = 2$, the competitive ratio of our algorithm is approximately 4.5911, which is almost twice as good as the best that can be done deterministically. We also give evidence that our algorithm is optimal for $w > 2$.

A similar problem, known as layered graph traversal, has been studied by Papadimitriou and Yannakakis [9] and Fiat, Foster, Karloff, Rabani, Ravid, and Vishwanathan [6]. Layered graph traversal is similar to the cow-path problem, but allows shortcuts between paths without going through the origin, and when exploring one path, 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 considering randomized algorithms, 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 [6].

2 Definitions

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

$$\text{cost}(\mathcal{A}, g) \leq c\ \text{dist}(g) + d,$$

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, $\text{cost}(\mathcal{R}, g)$ 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,

$$E[\text{cost}(\mathcal{R}, g)] \leq c\ \text{dist}(g) + d,$$

where c and d are constants as before.

```
σ ← A random permutation of {0, 1, 2, ···, w − 1};
ε ← A random real uniformly chosen from [0, 1);
d ← r^ε;
p ← 0;
repeat
        Explore path σ(p) up to distance d;
        if goal not found then return to origin;
        d ← d · r;
        p ← (p + 1) mod w;
until goal found;
```

Figure 1: Algorithm SmartCow

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 $0, 1, \cdots, w-1$. The general outline of SmartCow can be found in Figure 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.

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.

THEOREM 3.1. *For any fixed $r > 1$, Algorithm* SmartCow *has competitive ratio*

$$1 + \frac{2}{w} \cdot \frac{1 + r + r^2 + \cdots + r^{w-1}}{\ln r}.$$

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. 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 Figure 1 that SmartCow proceeds in stages, where at stage $i \in \{0, 1, 2, \cdots\}$ the algorithm sweeps distance $r^{i+\epsilon}$ 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 $m = c \geq k + 1$

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

and the expected value is easily calculated as

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

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

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

Thus, the resulting expected distance traveled in this case is

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

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

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

In this case, $E[r^{\epsilon}|F]$ and $E[r^{\epsilon}|\bar{F}]$ can be found as follows:

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

$$E[r^{\epsilon}|\bar{F}] = \int_1^{r^{\delta}} x \cdot \frac{1}{\text{Prob}(\bar{F}) x \ln r}\, dx = \frac{r^{\delta} - 1}{\text{Prob}(\bar{F})\ln r}.$$

Using these values, $E[D|m = k]$ is equal to

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

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

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

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

Therefore, the above equation can be expanded to

$$\begin{aligned} E[D] = {} & \\ & \frac{2}{w(r - 1)\ln r}\left[(r - r^{\delta})(r^k - 1) + (r^{\delta} - 1)(r^{k+w} - 1)\right] + \\ & \qquad \frac{n}{w} + \sum_{i=k+1}^{k+w-1}\left[\frac{2(r^m - 1)}{w \ln r} + \frac{n}{w}\right] \\ = {} & \frac{2}{w(r - 1)\ln r}\left[(r - r^{\delta})(r^k - 1) + \right. \\ & \left. (r^{\delta} - 1)(r^{k+w} - 1) + (r - 1)\sum_{i=k+1}^{k+w-1}(r^m - 1)\right] + n \\ = {} & \frac{2}{w(r - 1)\ln r}\left[r^{k+\delta}(r^w - 1) - w(r - 1)\right] + n \\ \leq {} & \left[\frac{2(r^w - 1)}{w(r - 1)\ln r} + 1\right] n. \end{aligned}$$

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

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

which is exactly the ratio 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, Culberson, and Rawlins [2]. For example, when $w = 2$ their algorithm has a competitive ratio of 9, while Theorem 3.1 states that SmartCow has competitive ratio $1 + \frac{1+r}{\ln r}$. 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 deterministic 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 of the famous von Neumann minimax principle [12]. In particular, we will 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 lemma 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 $\epsilon > 0$. We will denote a particular distribution function by $f_{\epsilon,w}$, and we will use $\mathrm{Opt}(\epsilon, w)$ to denote the optimal competitive ratio of any deterministic algorithm with input distribution $f_{\epsilon,w}$. Our goal will be to show that $\lim_{\epsilon \to 0} \mathrm{Opt}(\epsilon, w)$ exists, and give a value for this limit. The following lemma shows how this is relevant.

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

Proof. For the sake of contradiction, assume that there is a randomized cow-path algorithm that achieves competitive ratio $\rho < \ell$. Let $\delta = \frac{\ell - \rho}{2}$. Now by the formal definition of the limit, there exists an ϵ_0 such that for all $\epsilon < \epsilon_0$, $|\mathrm{Opt}(\epsilon, w) - \ell| \leq \delta$. In other words, for any $\epsilon < \epsilon_0$,

$$\mathrm{Opt}(\epsilon, w) \geq \ell - \rho = \frac{2\ell - (\ell - \rho)}{2} = \frac{\ell + \rho}{2} > \rho.$$

But, by Yao's lemma, this implies that $\mathrm{OptR}(w) > \rho$, a contradiction with the original assumption that there exists a randomized algorithm with competitive ratio ρ. ∎

Now we define the density function $f_{\epsilon,w}$. 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 ϵ, the path is chosen uniformly from all possible paths. Thus, we will use $f_{\epsilon,w}$ to denote only the distance down the chosen path to the goal. The density function we use is

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

Any deterministic algorithm can be defined by a sequence $(s_0, p_0), (s_1, p_1), \cdots, (s_k, p_k), \cdots$, 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 $p_k = (k \bmod w)$, and then the algorithm is completely specified by the sequence $s_0, s_1, \cdots, s_k, \cdots$. 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 $s_0, s_1, \cdots, s_k, \cdots$. For input distribution $f_{\epsilon,w}$, the expected competitive ratio of A is*

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

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

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

$$C_i \text{ is true iff } \begin{cases} 1 \leq D \leq s_i \text{ and } P = i \\ \qquad\qquad \text{when } 0 \leq i < w; \\ s_{i-w} < D \leq s_i \text{ and } P \equiv i (\mathrm{mod}\, w) \\ \qquad\qquad \text{when } i \geq w. \end{cases}$$

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

$$E[R] = \sum_{i=0}^{\infty} p_i E[R|C_i]. \tag{4.1}$$

In computing the expected values $E[R|C_i]$ 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 $E[R|C_i]$, 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

$$E[R|C_i] = E\left[\sum_{j=0}^{i-1} \frac{2s_j}{D} + 1 | C_i\right] \tag{4.2}$$

$$= E[\frac{1}{D}|C_i]\sum_{j=0}^{i-1}(2s_j)+1.$$

To calculate $E[\frac{1}{D}|C_i]$, 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,

$$\begin{aligned} E[\frac{1}{D}|C_i] &= \int_{s_{i-w}}^{s_i} \frac{1}{x}\frac{\epsilon}{wp_i}x^{-(1+\epsilon)}dx \\ &= \frac{\epsilon}{wp_i(1+\epsilon)}\left(s_{i-w}^{-(1+\epsilon)} - s_i^{-(1+\epsilon)}\right). \end{aligned}$$

Combining this with equation (4.2) gives the conditional expected competitive ratio.

Summarizing all cases for the conditional expectation,

$$E[R|C_i] = \begin{cases} 1 & \text{if } i = 0; \\ \dfrac{2\epsilon}{wp_i(1+\epsilon)}\left(1 - s_i^{-(1+\epsilon)}\right)\displaystyle\sum_{j=0}^{i-1}s_j + 1 & \text{if } 1 \le i < w; \\ \dfrac{2\epsilon}{wp_i(1+\epsilon)}\left(s_{i-w}^{-(1+\epsilon)} - s_i^{-(1+\epsilon)}\right)\displaystyle\sum_{j=0}^{i-1}s_j + 1 & \text{if } i \ge w. \end{cases}$$

Combining these results with equation (4.1) shows that (after some algebraic manipulation) $E[R]$ is

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

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 $s_0, s_1, \cdots, s_k, \cdots$ define the optimal deterministic algorithm for a fixed ϵ, and let $\mathrm{Opt}(\epsilon, 2)$ denote the competitive ratio given in Lemma 4.2. Rewrite this formula in cleaner form for $w = 2$:

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

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

$$\begin{aligned} R(\epsilon) &= \sum_{i=0}^{\infty}\frac{s_i + s_{i+1}}{s_i^{1+\epsilon}} = \frac{s_0 + s_1}{s_0^{1+\epsilon}} + \sum_{i=1}^{\infty}\frac{s_i + s_{i+1}}{s_i^{1+\epsilon}} \\ &= 1 + s_1 + \sum_{i=1}^{\infty}\frac{s_i + s_{i+1}}{s_i^{1+\epsilon}} \end{aligned}$$

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

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

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

But this is easily lower bounded by

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

In other words,

$$R(\epsilon) \ge \frac{1+s_1}{1-s_1^{-\epsilon}},$$

so

$$\mathrm{Opt}(\epsilon, 2) \ge 1 + \frac{\epsilon}{1+\epsilon}\left(1 + \frac{1+s_1}{1-s_1^{-\epsilon}}\right).$$

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 as a lower bound above. 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,

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

By Lemma 4.1, we know that $\mathrm{OptR}(2) \ge \lim_{\epsilon\to 0}\mathrm{Opt}(\epsilon, 2)$, so we can bound $\mathrm{OptR}(2)$ by

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

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

Unfortunately, we have been unable to make this proof method work for $w > 2$. We conjecture that SmartCow is optimal for all values of w, but suspect that a proof of this fact must use a different approach than we used in the preceding theorem. Explicitly stated, we make the following conjecture.

CONJECTURE 4.1. *The optimal competitive ratio achievable by any algorithm for the w-lane cow-path problem is given by*

$$\min_{r>1}\left(1+\frac{2}{w}\frac{1+r+r^2+\cdots+r^{w-1}}{\ln r}\right).$$

Since this is exactly the ratio achievable by SmartCow, *algorithm* SmartCow (*with the appropriate minimizing r*) *is an optimal randomized algorithm.*

5 Minimizing the Competitive Ratio

Recall from Theorem 3.1 that algorithm SmartCow has competitive ratio

$$1+\frac{2}{w}\cdot\frac{1+r+r^2+\cdots+r^{w-1}}{\ln r}, \tag{5.1}$$

where r is a fixed algorithm parameter. In other words, 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 equation (5.1).

THEOREM 5.1. *The unique solution of the equation*

$$\ln r = \frac{1+r+r^2+\cdots+r^{w-1}}{r+2r^2+3r^3+\cdots+(w-1)r^{w-1}} \tag{5.2}$$

for $r > 1$, denoted by r_w^*, gives the minimum value for equation (5.1).

Proof. To minimize equation (5.1), we only need to minimize the part that depends on r. Call this function $f(r)$, where

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

This function is continuous, and $f(r)$ 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:

$$f'(r)=\frac{A-B}{(\ln r)^2},$$

where

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

and

$$B=(1+r+\cdots+r^{w-1}).$$

The denominator is non-zero and finite for all $r \in (1,\infty)$, and the numerator is zero exactly when equation (5.2) is true. In other words, the minimizing r must satisfy equation (5.2) — by showing that there is only one such r, we will have proved the theorem.

We need to show that equation (5.2) 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 equation (5.2) is monotonically *decreasing*, so it follows that equation (5.2) can have *at most* one solution. To see this, consider the right hand side of equation (5.2):

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

Taking the derivative with respect to r gives

$$g'(r)=\frac{C-D}{(r+2r^2+\cdots+(w-1)r^{w-1})^2},$$

where

$$C=r(1+2r+\cdots+(w-1)r^{w-2})^2$$

and

$$D=(1+r+\cdots+r^{w-1})(1+4r+\cdots+(w-1)^2r^{w-2}).$$

The denominator of $g'(r)$ 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 $g'(r)$ can be written as $\sum_{i=0}^{2w-2} c_k r^k$, where the coefficients c_k are

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

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

Now that we have shown that equation (5.2) 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,

w	r_w^*	Competitive Ratio of SmartCow	Optimal Deterministic Ratio
2	3.59112	4.59112	9
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

Table 1: Approximate values for small w.

since the function is continuous, it must have a root in the interval $(1, \infty)$. Thus we have proved that equation (5.2) has exactly one solution for $r > 1$. ∎

The value r_w^* can be found for any given w from equation (5.2) by 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 5, with the optimal deterministic ratio shown for reference.

Recall that by the results of Section 4, the ratio achieved for $w = 2$ (which is ≈ 4.59112 from the Table 5) is optimal for randomized algorithms. Furthermore, we believe that this method gives optimal randomized algorithms for $w \geq 3$ as well.

6 Conclusions

In this paper we have given a new randomized algorithm, SmartCow, for the cow-path problem. Furthermore, we have shown that for the problem with two paths 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 only obtained for $w = 2$ (showing that SmartCow is optimal for $w = 2$). It is an important open problem to determine a lower bound for randomized algorithms when $w \geq 3$ — this may be possible by minimizing the general form given in Lemma 4.2, or by entirely different means.

References

[1] J. A. Aslam and A. Dhagat. "Searching in the Presence of Linearly Bounded Errors." *STOC*, pp. 486–493, 1991.

[2] R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins. "Searching in the Plane." *Pre-print*, 1991.

[3] J. L. Bentley and A. C.-C. Yao. "An Almost Optimal Algorithm for Unbounded Searching." *Information Processing Letters*, Vol. 5, pp. 82-87, 1976.

[4] A. Blum, P. Raghavan, and B. Schieber. "Navigating in Unfamiliar Geometric Terrain." *STOC*, pp. 494–504, 1991.

[5] X. Deng and C. H. Papadimitriou. "Exploring an Unknown Graph." *FOCS*, pp. 355–361, 1990.

[6] A. Fiat, D. P. Foster, H. Karloff, Y. Rabani, Y. Ravid, and S. Vishwanathan. "Competitive Algorithms for Layered Graph Traversal." *FOCS*, pp. 288–297, 1991.

[7] A. Fiat, Y. Rabani, and Y. Ravid. "Competitive k-server Algorithms." *FOCS*, pp. 454–463, 1990.

[8] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley Publishing Company, Reading, MA, 1984.

[9] C. H. Papadimitriou and M. Yannakakis. "Shortest Paths without a Map." *ICALP*, pp. 610–620, 1989.

[10] R. L. Rivest, A. R. Meyer, D. J. Kleitman, and K. Winklmann. "Coping with Errors in Binary Search Procedures." *Journal of Computer and System Sciences*, Vol. 20, pp. 396–404, 1980.

[11] D. D. Sleator and R. E. Tarjan. "Amortized Efficiency of List Update and Paging Rules." *CACM*, Vol. 28, pp. 202–208, Feb. 1985.

[12] A. Yao. "Probabilistic Computations: Towards a Unified Measure of Complexity." *FOCS*, pp. 222–227, 1977.