Competitive Analysis

6.046 Recitation Notes

November 3, 2006

1 Outline

In this recitation, we used competitive analysis to analyze algorithms for two different problems:

- Ski Rental Problem
- Robot Search Problem

For each problem, we describe a competitive deterministic algorithm, and then show how introducing randomness into the algorithm can improve the competitive ratio.

2 Ski Rental Problem

2.1 Problem Setup

The ski rental problem arises from the following scenario. You want to start skiing, but you don't own any skis. On each day you go skiing, you can either choose to rent skis for cost 1, or you can buy skis for cost s. Since you don't know when you might get sick of skiing, (or when you might break your leg and be unable to ski again), the number of days d that you will end up skiing is unknown to you. For this problem, assume s is a positive integer bigger than 1. In examples, we will use $s = 10$.

We model the ski rental problem as online problem and analyze it using competitive analysis. For the ski rental problem, an *input sequence* σ represents a sequence of some number of requests to ski. An algorithm A describes some strategy for deciding when to rent skis and when to buy skis. For now, we consider only *deterministic* algorithms A; in this case, the space of all "reasonable" deterministic algorithms can be represented by a single integer marking when to buy skis. To describe σ and A more precisely, we use the following notation:

- Let σ_d denote the input sequence that corresponds to skiing for d days, and then stopping.
- Let A_i denote the deterministic algorithm that rents skis for the first $i-1$ days ,and then buys skis on day i.

Next, recall the definition of a (strictly) competitive online algorithm:

Definition 1. *Let $A(\sigma)$ denote the cost of an algorithm A on input sequence σ. Let $A^*(\sigma)$ denote the cost of an optimal, offline algorithm on input σ. An online algorithm A is* strictly α-competitive *if for any input sequence,*

$$A(\sigma) \leq \alpha A^*(\sigma).$$

In this case, α is said to be the competitive ratio *of algorithm A.*

Competitive analysis can be viewed as a game between the you, the algorithm designer, and a malicious adversary.

1. You choose an algorithm A.

2. The adversary chooses an input sequence σ which maximizes the ratio $\gamma = A(\sigma)/A^*(\sigma)$. The competitive ratio α of the algorithm A is the maximum γ over all possible inputs σ.

For the ski rental problem, you first choose an algorithm A_i, and then based on your choice of i, the adversary chooses the worst possible d, i.e., the one that maximizes $A(\sigma_d)/A^*(\sigma_d)$.

2.2 Intuition

In the ski rental problem, we can gain intuition about the problem by considering the extremes. Suppose we pick i to be large, i.e., we rent for a long time before buying. Then, the adversary will choose d to be large; then, we keep paying the cost of renting skis while the optimal offline algorithm has already purchased skis for cost s. On the other hand, if we choose a small i, i.e., we buy our skis early, then adversary will choose a small d; then, we pay s to buy the skis, but the optimal offline algorithm rents skis every time.

We achieve a balance between these two extremes by renting skis until we can amortize the cost of buying the skis against the cost of our previous rentals. Since buying skis costs $s = 10$ units, if we rent for $s - 1$ days, and then buy skis on day s, then even if adversary makes us stop skiing immediately after we buy the skis, we can show we never pay more than twice what the optimal, clairvoyant algorithm would have. In the next few sections, we formalize this intuition.

2.3 Computing the Competitive Ratio

For the ski rental problem, we are actually able to directly compute both $A^*(\sigma_d)$, the cost of the optimal algorithm on any input sequence σ_d, and $A_i(\sigma_d)$, the cost of algorithm A_i on σ_d.

Optimal Cost

The optimal, offline algorithm knows d, the number of days that you will go skiing. Thus, the optimal algorithm chooses the cheapest of two options: buy skis immediately for cost s, or rent on all d days for cost d. Thus,

$$A^*(\sigma_d) = \min\{s, d\}. \tag{1}$$

Figure 1 illustrates $A^*(\sigma_d)$ for $s = 10$.

Cost of A_i

Similarly, we can compute the cost $A_i(\sigma_d)$, the cost of the algorithm that buys skis on day i if we ski for d days. If $d < i$, then the algorithm A_i rent skis for a cost of d. If $d \geq i$, then A_i rents skis for $(i - 1)$ days and purchases skis on day i, for a total cost of $(i - 1) + s$. Thus,

$$A_i(\sigma_d) = \begin{cases} d & \text{if } d < i, \\ (i-1) + s & \text{if } d \geq i. \end{cases} \tag{2}$$

Figure 2 illustrates $A_6(\sigma_d)$ for $s = 10$.

Competitive Ratio for A_i

Let $\gamma(i)$ denote the competitive ratio for a fixed algorithm A_i. We compute $\gamma(i)$ by considering two cases for the algorithm A_i, first when $i \leq s$ and second when $i > s$.

If $i \leq s$, then the plot of the competitive ratio γ has the general shape shown in Figure 3. The ratio γ is maximized at $d = i$. The optimal choice of d for the adversary is to tell you to stop skiing on the same day you choose to buy skis. If the adversary waits and picks $d > i$, then the optimal cost keeps going up, while the cost of A_i has already maxed out. If the adversary picks $d = i$, the $\gamma = (i - 1 + s)/i = 1 + (s - 1)/i$.

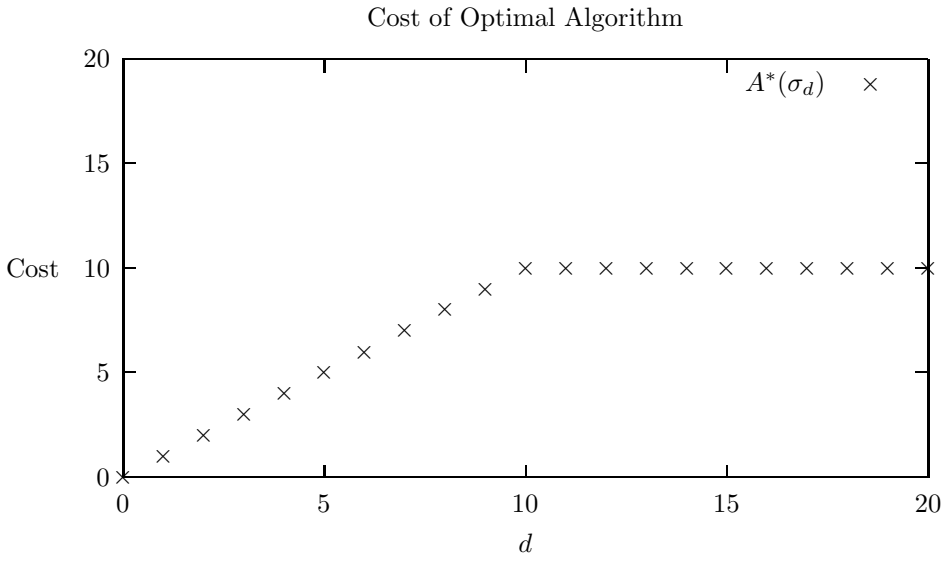

Figure 1: Plot of $A^*(\sigma_d)$, for $s = 10$.

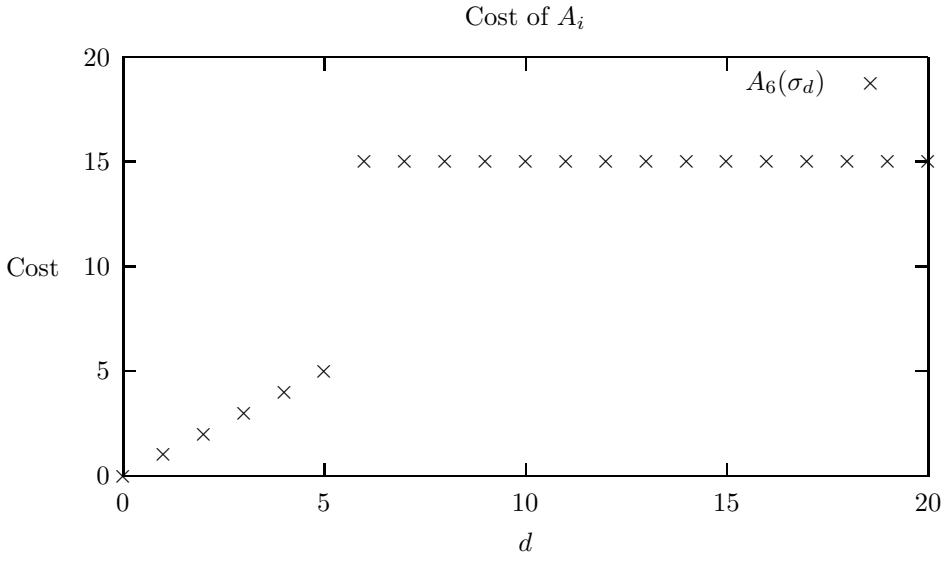

Figure 2: Plot of $A_6(\sigma_d)$, for $s = 10$.

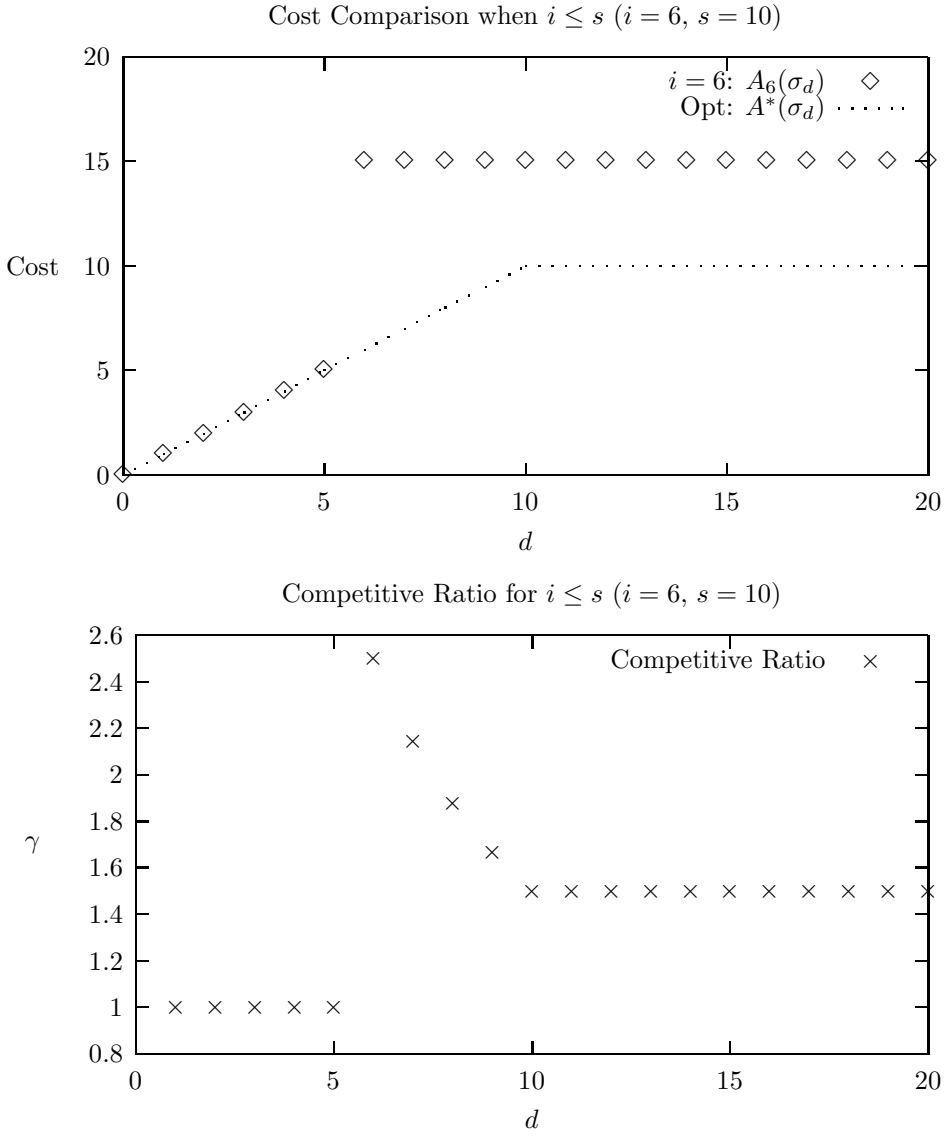

Figure 3: Plot of competitive ratio γ when $i \leq s$ ($i = 6, s = 10$).

On the other hand, if $i > s$, then the plot of γ has the general shape shown in Figure 4. The adversary can choose any $d \geq i$ to maximize the competitive ratio. In this case, $\gamma = (i - 1 + s)/s = 1 + (i - 1)/s$.

For any fixed value of i, the adversary can choose $d = i$ as the worst possible input σ_d for your algorithm. In this case,t he competitive ratio as a function of i is

$$\gamma(i) = \begin{cases} 1 + \frac{s-1}{i} & \text{if } i \leq s, \\ 1 + \frac{i-1}{s} & \text{if } i > s. \end{cases} \tag{3}$$

2.4 Optimizing the Competitive Ratio

For any fixed value of i, (3) gives us the competitive ratio on the worst possible input. Thus, to pick the best (deterministic) algorithm, we need to choose the value of i that minimizes the ratio $\gamma(i)$.

If we consider only $i \leq s$, then the optimal choice is $i = s$, giving us a competitive ratio of $2 - 1/s$. If we consider only $i > s$, then $i = s + 1$ is the optimal choice, giving us a competitive ratio of 2. Thus, the best of these options is to choose $i = s$, giving us the best competitive ratio of

$$\alpha = 2 - \frac{1}{s}.$$

Thus, for $s = 10$, the best competitive ratio we can achieve with a deterministic algorithm is $\alpha = 1.9$.

2.5 Randomized Algorithms for Ski Rental

Up until this point, we have only considered deterministic algorithms of the form A_i (buy skis on day i). A *randomized* online algorithm, however, might flip coins during its execution and use the outcomes of those flips to make decisions.

When analyzing a randomized online algorithm with an *oblivious* adversary, the game proceeds as follows:

1. You choose an algorithm A, which can make decisions based on the outcomes of coin flips.
2. The oblivious adversary chooses an input sequence σ to maximize the expected competitive ratio $\alpha = E[A(\sigma)]/A^*(\sigma)$.
3. Flip coins and observe the result.

Note that an oblivious adversary is unable to see the outcome of the coin flips. Intuitively, introducing randomness into the algorithm makes it more difficult for the adversary to elicit worst-case behavior for your algorithm sincethe adversary can not predict exactly what your algorithm will do.

For the ski rental problem, it turns out that by making a single coin flip, it is possible to achieve a competitive ratio better than $2 - 1/s$, the best ratio for any deterministic algorithm.

A Simple Randomized Algorithm for Ski Rental

Motivated by the optimal deterministic algorithm we computed in Section 2.4, we can construct a simple randomized algorithm which, with a single coin flip, achieves a better competitive ratio in expectation.

For simplicity, we consider only the case where s is an even integer that is at least 2. Consider the following algorithm $\widetilde{A}$ which flips a biased coin and chooses between two algorithms:

$$\widetilde{A} = \begin{cases} A_{s/2} & \text{with probability } p. \\ A_s & \text{with probability } (1 - p). \end{cases} \tag{4}$$

In other words, if the coin comes up heads, then $\widetilde{A}$ chooses to buy skis on day $s/2$; otherwise, the coin comes up tails, and $\widetilde{A}$ chooses to follow the optimal deterministic algorithm from before, which chooses to buys skis on day s. We want to choose the value of p which will minimize the expected competitive ratio.

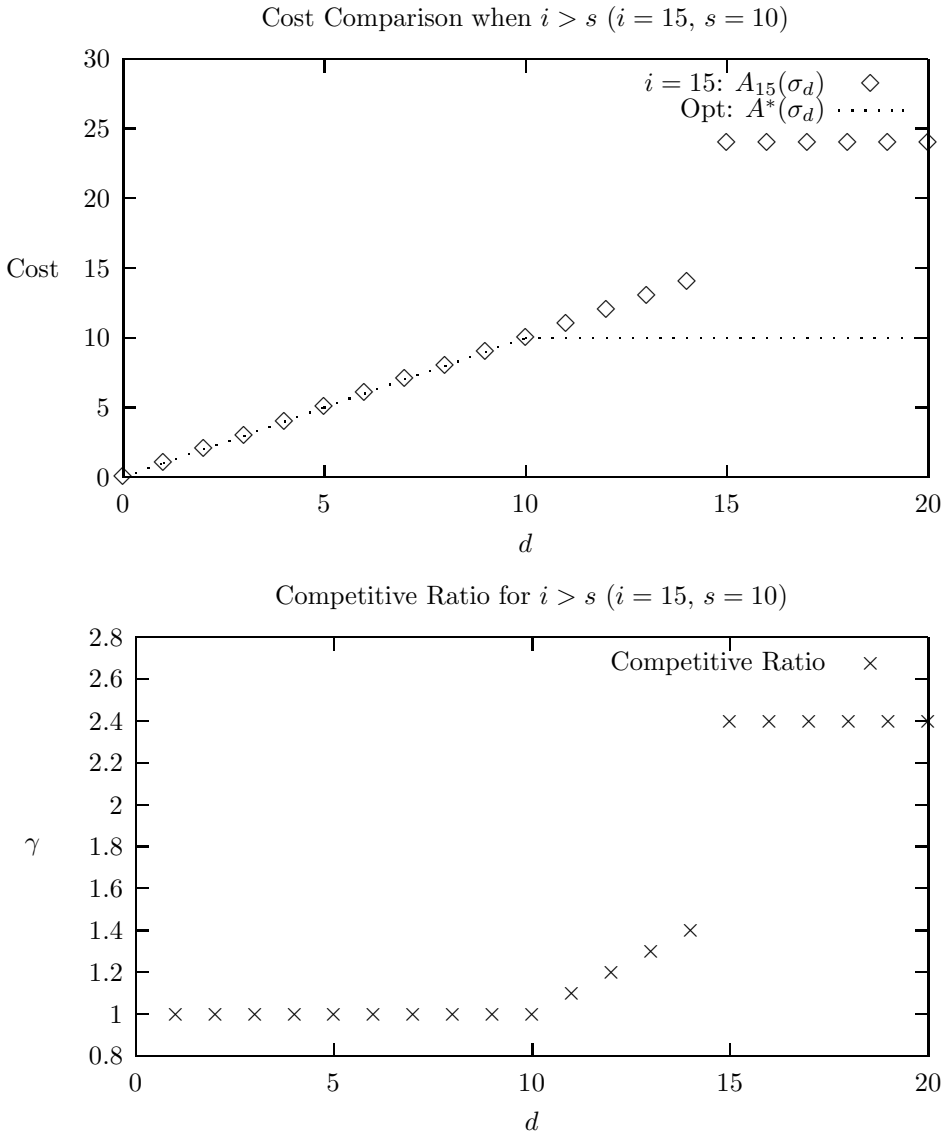

Figure 4: Plot of competitive ratio γ when $i > s$ ($i = 15, s = 10$).

Cost for $\widetilde{A}$

We compute $E[\widetilde{A}(\sigma_d)]$, the expected cost of $\widetilde{A}$ on an input sequence σ_d, by considering three cases for d.

1. $d < s/2$: In this case, we always rent skis, and $\widetilde{A}(\sigma_d) = d$.
2. $s/2 \leq d < s$: In this case, with probability p we buy skis on day $s/2$, and with probability $(1-p)$ we keep renting skis until we stop. Thus,
$$E[\widetilde{A}(\sigma_d)] = p\left(\frac{s}{2} - 1 + s\right) + (1-p)d.$$
3. $d \geq s$: In this case, with probability p we buy skis on day $s/2$, and with probability $(1-p)$ we buy our skis on day s.
$$E[\widetilde{A}(\sigma_d)] = p\left(\frac{s}{2} - 1 + s\right) + (1-p)\left(s - 1 + s\right).$$

The cost of the optimal offline algorithm A^* is the same as before; for cases 1 and 2, $A^*(\sigma_d) = d$, and for case 3, $A^*(\sigma_d) = s$. Thus, the competitive ratio γ as a function of d is:

$$E[\gamma] = \begin{cases} 1 & \text{if } d < s/2, \\ \frac{p}{d}\left(\frac{3s}{2} - 1\right) + (1-p) & \text{if } s/2 \leq d < s, \\ 2 - \frac{p}{2} - \frac{1}{s} & \text{if } d \geq s. \end{cases} \tag{5}$$

Optimizing $E[\gamma]$

Recall that in competitive analysis, we first choose the algorithm, and then the adversary chooses the input sequence. In this example, we choose p, and then the adversary chooses d. Thus, when analyzing the algorithm, we work backwards: first fix an algorithm (e.g., a p), and then determine what the adversary would do for each fixed algorithm (e.g., determine which value of d the adversary will choose to elicit the worst behavior for our fixed value of p). Then, we allow p to vary and choose the p that minimizes the competitive ratio.

From (5), for a fixed value of p, the adversary has three regions in which he can pick d. In first and last regions, $E[\gamma(d)]$ is constant; in the middle region, the adversary should pick $d = s/2$ to maximize the ratio. Thus, we can narrow down the adversary's choices to the largest of

$$E[\gamma(d)] = \begin{cases} 1 & \text{if } d < s/2, \\ 2p\left(1 - \frac{1}{s}\right) + 1 & \text{if } d = s/2, \\ 2 - \frac{p}{2} - \frac{1}{s} & \text{if } d \geq s. \end{cases} \tag{6}$$

From (6), it is clear that the adversary never chooses $d < s/2$, since we assumed $s > 1$, $d = s/2$ always gives a strictly larger ratio. Thus, the adversary chooses between the cases when $d = s/2$, and $d \geq s$.

Figure 5 shows how the expected competitive ratio changes as a function of p. The ratio when $d = s/2$ increases as p increases, while the ratio when $d \geq s$ decreases as p increases. By setting the two ratios equal to each other, we can determine where the point where the adversary is indifferent between choosing $d = s/2$ and $d \geq s$. This point occurs when

$$\begin{aligned} 2p\left(1 - \frac{1}{s}\right) + 1 &= 2 - \frac{1}{s} - \frac{p}{2} \\ p\left(\frac{5}{2} - \frac{2}{s}\right) &= 1 - \frac{1}{s} \\ p &= \frac{2}{5}\left(1 - \frac{1}{5s-4}\right). \end{aligned}$$

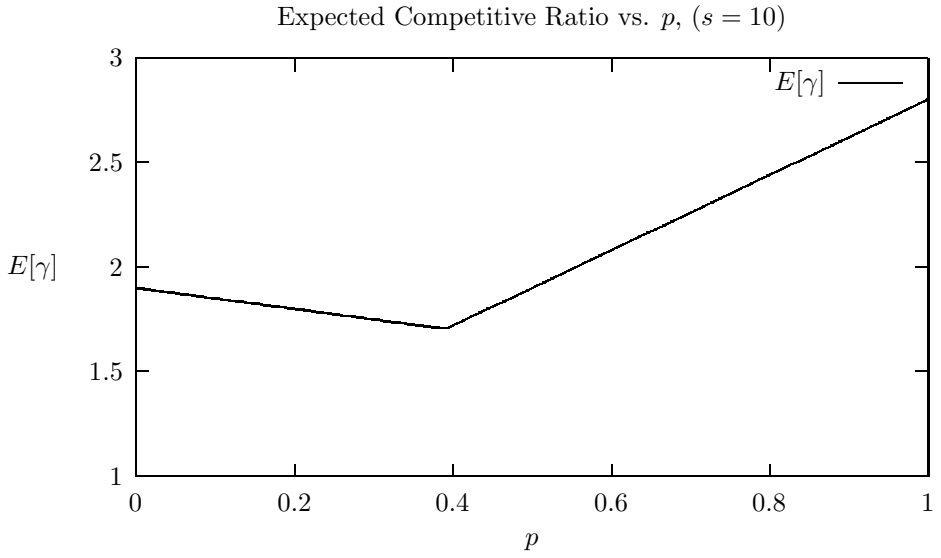

Figure 5: Plot of maximum value of $E[\gamma]$ from Equation (6), as a function of p, for $s = 10$.

From this analysis, we can conclude the best p to choose and compute the best expected competitive ratio for our algorithm $\widetilde{A}$.

$$p = \frac{2}{5}\left(1 - \frac{1}{5s-4}\right)$$
$$E[\gamma] = \frac{9}{5} - \frac{1}{s}\left(1 - \frac{s}{25s-20}\right).$$

Plugging in actual numbers, for $s = 10$, we have $p \approx 0.391$ and $E[\gamma] \approx 1.704$.

2.6 Notes

Note that our choice of randomized algorithm $\widetilde{A}$ was somewhat arbitrary, and is not necessarily the choice that minimizes the competitive ratio over all possible randomized algorithms for the ski rental problem.

One application of the ski rental problem is in the context of snoopy caches for multiprocessor systems [3]. In a continuous setting, [2] gives an optimal randomized algorithm for the ski rental problem which achieves a competetive ratio that approaches $e/(e-1) \approx 1.58$.

3 Robot Search Problem

Imagine we have a robot standing in front of an infinite 1-dimensional wall, at a position $x = 0$. You are told there exists a single door at some unknown position x^* along the wall. In particular, you do not know which direction the wall is in (i.e., whether $x^* > 0$ or $x^* < 0$). The goal of the robot search problem is to describe a good search algorithm for the robot for finding the door. For simplicity, we assume the door is not "too close" to the origin, meaning $|x^*| > 1$.

We also analyze this problem using competitive analysis. First, we choose a search algorithm, and then the adversary places the door at the worst possible location for our algorithm, i.e., the one that maximizes the ratio of the distance our algorithm walks to the distance the optimal algorithm walks. Since the optimal

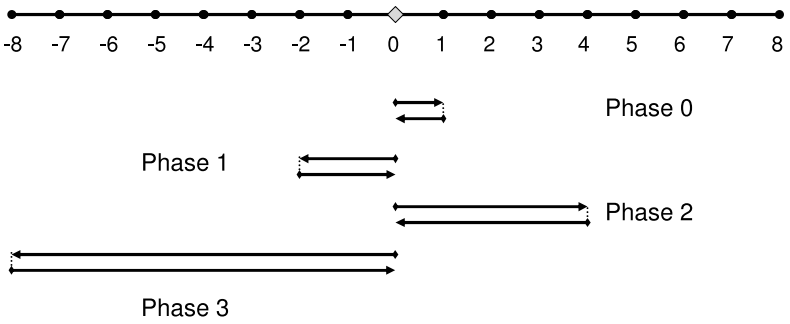

Figure 6: Search algorithm A_2 for the robot problem, illustrating the search performed in phases 0 through 3. During a phase i, the robot walks from 0 and $(-m)^i$ and back to 0, for a total distance in phase i of $2m^i$.

algorithm is offline, it knows exactly where the door is; thus, a robot programmed with the optimal algorithm walks a distance $|x^*|$.

One simple fact to notice is that our search algorithm must explore both regions $x > 0$ and $x < 0$; otherwise, the adversary places the door in the region the robot does not explore and the competetive ratio is infinite.

3.1 Deterministic Algorithm

Consider the following algorithm A_m, with $m > 1$, which searches for the door in phases. Number the phases starting at $i = 0$; in phase i, the robot starts at the origin, walks to $(-m)^i$ and walks back to the origin, stopping early only if it finds the door. In any phase i where the robot does not find the door, it walks a distance $2m^i$. Figure 6 illustrates the search algorithm in phases 0 through 3, for $m = 2$.

Only the absolute distance $|x^*|$ affects the cost of the optimal algorithm. Therefore, we divide the possible locations for the door into regions $1, 2, \ldots k$, where region k is the space satisfying $m^{k-1} < |x^*| \leq m^k$.

Suppose $|x^*|$ falls into region k. Then the robot did not find the door in phases 0 through $k-1$. If we are lucky, then $-(-m)^{k-1} < x^* \leq (-m)^k$, the robot finds the door in phase k, and the distance walked is

$$A_m(x^*) = \sum_{i=0}^{k-1} 2m^i + |x^*| = 2\left(\frac{m^k - 1}{m-1}\right) + |x^*|. \tag{7}$$

On the other hand, if we are unlucky, then the robot does not discover the door until phase $k+1$. In this case,

$$A_m(x^*) = \sum_{i=0}^{k} 2m^i + |x^*| = 2\left(\frac{m^{k+1} - 1}{m-1}\right) + |x^*|. \tag{8}$$

The adversary, of course, wanting to maximize our cost, will place the door on the side that makes us unlucky. Thus, the ratio will be

$$\gamma = \frac{2}{|x^*|}\left(\frac{m^{k+1} - 1}{m-1}\right) + 1.$$

Since we assumed $|x^*|$ falls into region k, the worst ratio for region k occurs if the adversary places the door at a distance $m^k + \epsilon$ from the origin. Thus, we have

$$\gamma < 2\left(\frac{m^2 - \frac{1}{m^{k-1}}}{m-1}\right) + 1.$$

The competitive ratio is thus bounded above by

$$\alpha = 2\left(\frac{m^2}{m-1}\right) + 1, \tag{9}$$

with γ approaching α as k goes to infinity.

By taking derivatives, one can show that $m = 2$ minimizes (9), giving us $\alpha = 9$.

3.2 Randomized Algorithm

From our analysis of the deterministic algorithm, we see that we got lucky when the robot started walking in the "good" direction at the beginning of the algorithm. Since the adversary can see our algorithm, however, he always chooses to put the door in the "bad" direction. Thus, a natural way to improve the competitive ratio of our algorithm is to flip a fair coin to decide which direction to start walking in. If the adversary is oblivious, then he doesn't know the outcome of the coin flip, and we now have an equal chance of getting lucky or unlucky.

Formally, consider an algorithm $\widetilde{A}_m$ which first flips a fair coin. If the coin comes up heads, the robot in all phases i walks from 0 to $(-m)^i$ and back. If the coin comes up tails, then the robot in all phases i walks from 0 to $-(-m)^i$ and back. By combining Equations (7) and (8) and dividing by $|x^*|$, our expected competitive ratio is

$$\begin{aligned}\gamma &= \frac{1}{2}\left(\frac{2}{|x^*|}\left(\frac{m^k-1}{m-1}\right)+1\right) + \frac{1}{2}\left(\frac{2}{|x^*|}\left(\frac{m^{k+1}-1}{m-1}\right)+1\right) \\ &= 1 + \frac{1}{|x^*|}\left(\frac{m^k+1+m^{k-1}-2}{m-1}\right).\end{aligned}$$

Choosing $|x^*| = m^k + \epsilon$ as before, and letting k approach ∞, we have that

$$\gamma < 1 + \left(\frac{m^2+m}{m-1}\right).$$

If we choose $m = 2$, the we get a competetive ratio of $\alpha = 7$. If we set $\frac{d\gamma}{dm} = 0$, we find that $m = 1+\sqrt{2} \approx 2.414$ minimizes the expression for γ, giving us a competitive ratio of $\alpha = 4 + 2\sqrt{2} \approx 6.8284$.

3.3 Notes

The robot search problem we described is sometimes referred to in the literature as a cow-path problem. In the w-lane cow-path problem (a cow stands at the origin and must search down w different paths to reach a field. The field lies a distance n away from the origin down one path, while all other paths continue forever. The robot search problem we described is a special case of $w = 2$. See [1] for a discussion of some of the relevant literature.

It is known that the optimal deterministic algorithm for the 2-lane cow-path problem has a competitive ratio of 9, meaning that the search algorithm with $m = 2$ is an optimal deterministic algorithm. In Kao, Reif, and Tate in [1] give a randomized algorithm for the 2-lane cow-path problem with competitive ratio of approximately 4.5911, and prove a matching lower bound that shows their algorithm is optimal. In addition to flipping a coin to randomly choose an initial direction, their algorithm randomly chooses a value for m, i.e., $m \approx 3.59112^{\epsilon}$, where ϵ is chosen uniformly from $[0, 1)$.

References

[1] Kao, Reif, and Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1993.

[2] Anna R. Karlin, Mark S. Manasse, Lyle A. McGeoch, and Susan Owicki. Competitive randomized algorithms for non-uniform problems. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 301–309, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.

[3] Anna R. Karlin, Mark S. Manasse, Larry Rudolph, and Daniel D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.