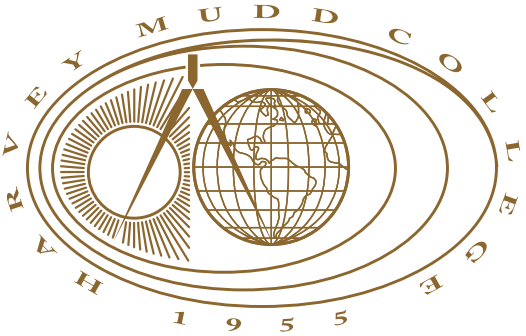

Searching Stars for a Moving Hider

Jennifer Iglesias

Ran Libeskind-Hadas, Advisor

Nicholas Pippenger, Reader

May, 2012

HARVEY MUDD
COLLEGE
Department of Mathematics

Abstract

In a search game, a seeker searches for a hider in some space. The seeker wishes to find the hider as quickly as possible, and the hider wishes to avoid capture as long as possible. In this paper, I will focus on the case where the search space is a star, and the only information the seeker has is the speed of the hider. I will provide algorithms for some cases where the seeker is guaranteed to find the hider and prove optimality for some of these cases. Also, I will look at some cases where the hider can avoid capture indefinitely. I will also present some results for searching on trees.

Contents

List of Figures

Acknowledgments

I would like to thank Professor Ran Libeskind-Hadas, for helping me with thesis this whole year, and listening to my problems even when they weren't thesis related. I would also like to thank my second reader, Prof. Nicholas Pippenger, for being patient with me. Lastly, I would like to thank the Mathematics Department in general, for being supportive and helpful throughout my years at Harvey Mudd College.

Chapter 1

Introduction

In a search game, a seeker searches for a hider in some space. In some versions of the problem, the hider is stationary, and other times the hider is allowed to move but has a maximum speed. The space in which the game takes place can also be varied. In all cases, the seeker is denied some information about the game, whether it is the hider's location or information on the search space itself. The goal of the *hider* is to avoid capture if possible, and if he can't avoid capture then to maximize the time until capture. In contrast, the *seeker* wishes to minimize the time until capture. More precisely, the hider wishes to minimize the competitive ratio as defined below.

Definition A *competitive ratio* is the asymptotic worst case ratio between the time it takes the seeker to find the hider, and the time it would take the seeker to catch the hider if the seeker had perfect information.

Multiple search spaces are investigated in Alpern and Gal (2002). In the first part of the book, the authors investigate the case that the hider is stationary. Then they go on to look at when the hider is moving on a line, and then when the hider is moving in a compact space and the seeker has a radius of vision. In Papadimitriou and Yannakakis (1991), the case of traversing an unknown two-layered graph efficiently is explored. This can also be thought of as searching for a stationary hider on a two layered graph. Also, searching for a stationary hider on a line is a special case of traversing a two layered graph problem. In Kao et al. (1996), they prove the optimality of a randomized algorithm for searching for a stationary hider on a set of rays which all emanate from a common point. This particular problem of searching for an immobile hider on a set of rays emanating from a point is often referred to as the cow-path problem. The optimal strategy

employed for solving a cow-path problem is to visit each ray in order, going out further on each ray. By randomizing the order the legs are visited in and the ratio of how far in the seeker goes on each leg, then the expected value of the competitive ratio is optimized. In von Stengal and Werchner (1997), the search spaces are extended to general graphs with one seeker and an immobile hider.

This paper investigates what the best strategies are for the hider and the seeker in various acyclic graphs under the assumption that the hider can move, and there is only one seeker. These search spaces include a line, finite stars, infinite stars, and infinite trees in which all vertices have constant degree. (Here infinite means that the rays can extend infinitely from a node, not that there can be infinitely many branches). These cases are different from the compact spaces in that the hider could potentially get arbitrarily far away from the seeker. The structure of a finite star alone provides an interesting case, even though this is a finite space.

Throughout this paper, we assume that the seeker moves with speed 1 and the speed of the hider will always be $w < 1$. If $w \geq 1$ then the hider could run away from the seeker in an infinite space and never be caught, and we will look at strictly deterministic strategies. To avoid confusion, the hider will be referred to using the pronoun *he*, and the seeker will be referred to with the pronoun *she*. (To help remember this, note that hider and seeker start with the same letter as their assigned pronouns). The general information model we will use will be the seeker only knows the hider's maximum speed, while we allow the hider to know the seeker's current position and all future moves. Occasionally, we will deviate from this information model but we will make the new information model clear.

Chapter 2

Searching on a Line

For this section, the search space will be a line. Also, we assume the hider and seeker will start at least one unit apart. Let the starting point of the seeker be the origin.

From Alpern and Gal (2002), we know the best strategy for the hider is move away from the seeker at his maximum speed. Let the origin be the starting point of the seeker. The seeker obtains her best competitive ratio when she alternately moves east and west, and each time she passes the origin she goes a factor of 2 $\left(\frac{1+w}{1-w}\right)$ farther away from the origin than she did the last time she passed the origin. The seeker continues these oscillations around the origin until she catches the hider.

There was no proof in the literature that showed that the hider's best strategy is to always move away from the seeker. Although it may seem obvious, this conjecture merits proof. The following lemma is a result of this uncertainty.

Lemma 2.1. *Even if the hider has perfect information on the seeker (knows exactly where the seeker will be at every point in time from the start), his best strategy is to move away.*

Proof. Let the hider have some strategy that maximizes the seeker's competitive ratio. Consider the first time at which capture could occur if the seeker also had perfect information, call this time t. After time t, the hider wants to evade capture as long as possible. Therefore the hider will just move away from the seeker after this point in time. If the hider just moves away from the seeker from the start, then the time until capture is $d/(1-w)$ where d is the distance between the hider and seeker at the start. In this case, t can be determined by the starting point of the hider, and can achieve

any nonnegative value. Therefore the strategy in which the hider simply moves away from the seeker, must achieve a competitive ratio at least as large as any other strategy. So, moving away from the seeker is an optimal strategy for the hider. □

What should the hider do if he has no information on the seeker? One possible strategy for the hider is to pick a direction and move in that direction, hoping that the direction is away from the seeker. What should the seeker do in the case that the hider chooses incorrectly? In other words, we want to know what the seeker will do if the hider is moving toward him, and the seeker knows the hider is moving towards him. We will now temporarily work in the information model where the seeker knows the hider is moving towards her.

Due to the asymptotic nature of the competitive ratio, if the seeker stays put, or oscillates back and forth by at most a constant amount, then her competitive ratio is simply $\frac{1+w}{w}$. Under the assumption that the best strategy requires the seeker to increase the distance he searches in each direction by some ratio r. We hope that $r > 1$, otherwise this would only tell the seeker what to do for a finite amount of time, or tell him to oscillate back and forth a constant amount. In the worst case, the seeker would just barely miss the hider on one oscillation. Let's say that the seeker went out to r^n and the hider was ϵ away from the seeker at this point. The amount of time it took the seeker to get here is $\frac{2r^{n-1}}{1-\frac{1}{r}} + r^n$. Therefore the starting distance between the hider and seeker was

$$(\frac{2r^n}{r-1} + r^n)w + r^n.$$

Once the seeker barely misses finding the hider, the seeker will travel to $r^{n+1} + r^n$ in the other direction and then turn around. Therefore the total distance the seeker and hider travel in this time is $2(r^{n+1} + r^n)$. The time from almost finding the hider until the time until the seeker finds him is

$$\frac{2(r^{n+1} + r^n)}{w+1}.$$

Therefore the total time until capture is

$$\frac{2(r^{n+1} + r^n)}{w+1} + \frac{2r^n}{r-1} + r^n.$$

So our competitive ratio is

$$\begin{aligned}
\frac{\frac{2(r^{n+1}+r^n)}{w+1}+\frac{2r^n}{r-1}+r^n}{(\frac{2r^n}{r-1}+r^n)w+r^n}(w+1) &= \frac{2(r^{n+1}+r^n)+\frac{2r^n(w+1)}{r-1}+r^n(w+1)}{(\frac{2r^n}{r-1}+r^n)w+r^n} \\
&= \frac{2(r^{n+1}+r^n)(r-1)+2r^n(w+1)+r^n(w+1)(r-1)}{2r^nw+r^nw(r-1)+r^n(r-1)} \\
&= \frac{2r^{n+2}-2r^{n+1}+2r^{n+1}-2r^n+w2r^n+2r^n+wr^{n+1}-wr^n+r^{n+1}-r^n}{2wr^n+wr^{n+1}-wr^n+r^{n+1}-r^n} \\
&= \frac{2r^{n+2}+r^{n+1}-r^n+wr^n+wr^{n+1}}{wr^n+wr^{n+1}+r^{n+1}-r^n} \\
&= \frac{2r^2+r-1+w+wr}{w+wr+r-1} \\
&= 1+\frac{2r^2}{w+wr+r-1}.
\end{aligned}$$

Now we want to minimize this. Taking the derivative with respect to r we get

$$\frac{4r(w+wr+r-1)-2r^2(w+1)}{(w+wr+r-1)^2}.$$

We want to find when this is zero, or when $4r(wr+w+r-1)-2r^2(w+1)=0$. This is just a quadratic, but we can immediately factor r out. So $r=0$ may be a possible solution. Rewriting the rest nicely we get

$$\begin{aligned}
4wr+4w+4r-4-2wr-2r &= 0 \\
2wr+4w+2r-4 &= 0 \\
(w+1)r &= 2-2w \\
r &= 2\left(\frac{1-w}{1+w}\right).
\end{aligned}$$

The optimal ratio if the hider is moving away from the seeker with speed w is $2\frac{1+w}{1-w}$. Therefore simply changing the sign on w gives the desired optimal ratio if the hider is moving toward the seeker. When $w<1/3$, then the ratio is greater than one as desired. This would make it seem that oscillating back and forth is not any better than staying put when $w\geq 1/3$. One might conjecture that the case considered here is not actually the worst case for the seeker.

However, this case is indeed the worst case. If the hider and seeker met later on in this iteration of the seeker's oscillation, then they spent all that

extra time moving towards each other. Let the seeker be x farther away in the start than the case described. Then the time until first possible capture increases by $\frac{x}{1+w}$. The time until actual capture increases by $\frac{x}{1+w}$. Since our ratio is always greater than one, this always makes the ratio smaller. Therefore this case is actually the worst case.

Now imagine the hider has no information on the seeker's position. He could choose a direction and then force the worst ratio if he were moving away half the time. We need to check that if the hider is moving towards the seeker then the competitive ratio doesn't get worse. First, we need to calculate the ratio if the seeker is moving away from the hider. We know the optimal solution to this is to increase the distance the hider goes on each oscillation by a ratio of $2\frac{1+w}{1-w}$. Alpern and Gal give that this ratio is $\frac{1}{1-w}(1+8\frac{1+w}{(1-w)^2})$.

Now we will find the competitive ratio if the seeker is searching with ratio $r = 2\frac{1+w}{1-w}$, and the hider is moving toward the searcher with speed w. The competitive ratio found above is $1+\frac{2r^2}{w+wr+r-1}$. Plugging in our r we get

$$\begin{aligned} 1+\frac{2r^2}{w+wr+r-1} &= 1+\frac{2(2\frac{1+w}{1-w})^2}{w-1+(w+1)(2\frac{1+w}{1-w})} \\ &= 1+(\frac{1}{1-w})\frac{8(1+w)^2}{2(1+w)-(1-w)^2}. \end{aligned}$$

To compare these two ratios, we can see that $\frac{1}{1-w} > 1$ when $0 \leq w < 1$. Now we need to compare $\frac{1}{1-w}(8\frac{1+w}{(1-w)^2})$ and $(\frac{1}{1-w})\frac{8(1+w)^2}{2(1+w)-(1-w)^2}$. We can get rid of the factor of $\frac{8(1+w)}{1-w}$. So now we just have $\frac{1}{(1-w)^2}$ and $\frac{1+w}{2(1+w)-(1-w)^2}$ to compare. $\frac{1}{(1-w)^2}$ is always greater than one when $0 \leq w < 1$. $\frac{1+w}{2(1+w)-(1-w)^2}$ is only greater than one if $2(1+w)-(1-w)^2 < 1+w$. Rearranging, we need that $1+w < (1-w)^2$. The left-hand side is always greater than 1, and the right-hand side is always less than 1. Therefore $\frac{1+w}{2(1+w)-(1-w)^2}$ is always less than 1. Therefore we have that $\frac{1}{1-w}(1+8\frac{1+w}{(1-w)^2})$ is greater than $1+(\frac{1}{1-w})\frac{8(1+w)^2}{2(1+w)-(1-w)^2}$ for all $0 \leq w < 1$. The competitive ratio is not worse in the case when the hider is moving toward the seeker and the seeker assumes the hider is moving away from the seeker. Thus the optimal strategy for the hider is to pick a direction at random, and the optimal strategy for the seeker is to assume the hider is moving away.

Chapter 3

Searching on Stars

Next we consider a variant on the cow-paths problem. Specifically, we investigate the problem of searching for a hider on a star. The star has a single vertex of degree greater than two. In the case of finite stars, all edges emanating from the center vertex are of length one. For this section, we will return to the information model mentioned in the introduction, where the hider has complete information on the seeker, and the seeker only knows the maximum speed of the hider.

It may seem that the hider would just want to move away from the seeker in a star with infinite legs. The hider could also avoid capture by staying close to the center vertex, and swapping which path he was on whenever the seeker came close to the center. This makes the seeker's job more complicated as she must search the space and constantly check that the hider is not close to the center. It is possible that the seeker is not guaranteed capture even if the hider has no information. For example, the hider could search for the center, then change which path he is on periodically. This could possibly cause the hider and seeker to always be on different legs, so the hider couldn't be caught.

Even if the hider tries to avoid capture by either hiding at the center or moving away from the center, the seeker can still guarantee to catch the hider in certain cases. For example, if the hider chooses to hide close to the center and the seeker moves shorter distances away from the center on each subsequent leg she checks, then the hider may be forced to converge to the center also, or be caught sooner.

For this section, we will present an optimal algorithm for the seeker on the 3-star and prove optimality. We will then give three increasingly complex algorithms for the seeker on general stars. We will then prove

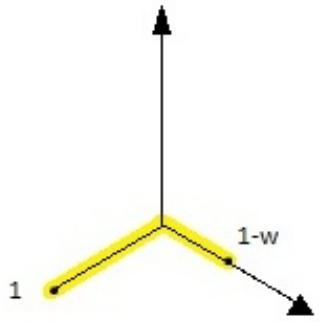

Figure 3.1 The possible starting positions of the hider.

optimality for the last algorithm. A key part of both proofs is the length cleared by the seeker. The cleared length is any area where the hider could not possibly be. The uncleared length is any area the hider can be.

Algorithm 3.1. *The algorithm for the seeker to find the hider will first be given on a star with three legs as this particular case has a unique proof method. The seeker will be guaranteed to find the hider if the hider's speed is less than 1/3. The seeker first travels to the end of the leg he starts on, and then back to the center (if the seeker starts at the center, then he can choose any leg to do this on). Then the seeker chooses a second leg, and go to the end of this leg and back to the center. Now she alternates searching these two legs, traveling in only as far in as the hider could possibly be. So if she traveled x far in on the last leg she searched then she travels $\frac{2xw}{1-w}$ in on the next leg she searches.*

This continues until the seeker converges to the center. The seeker will converge to the center since $w < 1/3$, which means $\frac{2xw}{1-w} < x$. Once the seeker converges to the center, then the hider can only be on the last remaining leg. So the seeker simply moves from the center to the end of this leg.

We will now show that Algorithm 3.1 is optimal for a finite star with three legs. While all of the legs are assumed to be of length 1, for the sake of analysis we begin by assuming that two of the legs are infinitely long as shown in Figure 3.1.

Let the seeker start at the origin, and let the hider start anywhere on the finite leg, and anywhere on one of the infinite legs within $1 - w$ of the origin. The possible positions of the hider are denoted by the shaded area

in Figure 3.1. Now the most area that the seeker can clear, if the hider can possibly be on more than one leg is $x(\frac{1}{2} - \frac{3w}{2})$, where x is the amount of time the hider spends on one leg. She can clear an area of $\frac{x}{2}$ on his way out on the leg. During the time the seeker is checking another leg, the uncleared length can extend by xw. On the leg the seeker is checking, the uncleared length can extend by at least $xw/2$. So the rate of clearing is upper bounded by $\frac{1}{2} - \frac{3w}{2}$.

If we clear the possible area on the infinite legs where the hider could be present first, we get that it takes time at least $1 + \frac{2-2w}{1-3w}$. If we check the finite leg before the end, then it will take time at least $2 + \frac{2+6w}{1-3w}$. Therefore we want to check the finite leg last. Since the max area the hider can be not on the finite leg is $1 - w$, then the seeker never has to go further out on a leg than 1 unit. If we restrict the space to be a finite 3-star where the hider can start anywhere, and each leg has length 1, then the same bound still holds, as the area which the hider could possibly be has not been restricted at all, and has actually extended.

Algorithm 3.1 achieves this bound if the seeker starts at the center. The time until capture can be expressed as

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

Note that Algorithm 3.1 takes two more units than the bound given. This is due to one leg starting off cleared in the special case which we used to derive the bound. Starting from the center it takes two units of time to clear a leg, and so we can't have done better. So this algorithm is optimal.

We will now generalize this algorithm to n-stars. Throughout the year, multiple different algorithms have been developed which guarantee capture for some w. These algorithms get progressively better, in both the range of w and the last point in time until capture. They also get progressively more complicated. Interestingly, in the case $n = 3$ all the subsequent algorithms result in the optimal algorithm given previously.

Algorithm 3.2. *This first algorithm works when $w < \frac{1}{2n-3}$. Label the legs as $\ell_1, \ell_2, \ldots, \ell_n$. We will assume the seeker starts at the center. The seeker will visit each leg except ℓ_n in order and goes to the end of the leg and back to the center. After this she will continue visiting the legs in order and skipping ℓ_n, going only in as far as the hider could possibly be on that leg and then returning to the center. Once she converges to the center, then the seeker knows the hider must be on ℓ_n and can simply go to the end of ℓ_n.*

We will now compute the time until the last possible capture of the seeker. The first pass through the legs takes time $2n-2$. At this point, the legs besides ℓ_n have a length of $(2n-4)w + (2n-2)w + \cdots + 2w$ where the hider could be. The seeker will clear $x/2$ if he travels x on a leg. While he travels x, the leg he is on will gain $xw/2$ length the hider can be, and the other legs will each gain xw length where the hider can be. So the rate at which the seeker clears is

$$\frac{1}{2} - \frac{w}{2} - (n-2)w = \frac{1}{2}(1-(2n-3)w).$$

Here we can see why this only works when $w \frac{1}{2n-3}$; otherwise the rate of clearing would be zero. So the time it takes Algorithm 3.2 to guarantee capture is

$$2(n-1) + \frac{2w(n-2)(n-1)}{2\frac{1}{2}(1-(2n-3)w)} + 1 = 2n - 1 + \frac{2w(n-2)(n-1)}{1-(2n-3)w}.$$

The second algorithm developed was based on the above computation. The seeker can increase the rate of clearing by not trying to clear all the legs at once. Unfortunately, this second algorithm still only works when $w < \frac{1}{2n-3}$, but guarantees capture earlier than the previous algorithm.

Algorithm 3.3. *We assume the seeker starts at the center and that the legs are labeled $\ell_1, \ell_2, \ldots, \ell_n$. This algorithm will go through n iterations; at the end of the ith iteration the area cleared is exactly the first i legs. At the beginning of the ith iteration, the seeker will go to the end of the ith leg and back. Now she will clear the first i legs. Note that the first $i-1$ legs were cleared when she started this process. The clearing can be done simply by picking any of the first i legs, going in on the leg as far as the hider could possibly be, and then returning to the center. This clearing process will eventually converge to the center. When she finishes the nth iteration, there is no where the hider could be, so he must have been caught somewhere during the process.*

The time for Algorithm 3.3 to guarantee capture is much more complicated as the clearing rate changes at each iteration. The rate of clearing on the ith iteration where i is not 1 or n is $1/2 - w/2 - (i-1)w$. After the first leg is cleared in the ith iteration there is $2(i-1)w$ length to clear on those legs. So the time it takes to run this algorithm is

$$2n - 1 + \sum_{i=2}^{n-1} \frac{2(i-1)w}{1/2 - w/2 - (i-1)w} = 2n - 1 + \sum_{i=1}^{n-2} \frac{4iw}{1-w-2iw}.$$

This sum can not be simplified as it is harmonic. Note that each term in the sum can be upper bounded by $\frac{4(n-2)w}{1-(2n-3)w}$, therefore Algorithm 3.3 runs faster than Algorithm 3.2.

Algorithm 3.4. *This algorithm works when $w < \frac{1}{n}$ when n is odd, and $w < \frac{1}{n-1}$ when n is even. We will assume that the seeker starts at the center. Let L be the set of all n legs and let S be a set of legs which starts out empty. The seeker first chooses a leg, adds the leg to S, and fully searches that leg. Now as long as the number of legs in S is less than $n/2$, she chooses another leg ℓ and adds it S. Now the seeker fully searches this leg, and then clears the parts of legs in S which might now contain the seeker. The clearing can be done simply by picking any leg in S the hider could be on, going out on the leg as far as the hider could possibly be, and then returning to the center. This clearing process will eventually converge to the center. At this point the seeker then repeats this process given that S has size less than $n/2$.*

Once S has size $n/2$, the seeker's strategy changes. Now consider the legs in $L - S$, let $|L - S| = i$. The hider could be anywhere on these legs. Now consider the steps taken for when there are i legs in S. If the seeker does the clearing process in reverse on the legs in $L - S$, then she will clear exactly one leg and the hider could be anywhere on the remaining legs and nowhere on the legs in S. This clearing process will have the seeker moving tiny amounts away from the center at first, and gradually the seeker searches further in on each leg in $L - S$. Eventually, the hider will clear enough length on the legs in $L - S$ that she can fully search her chosen leg without the hider getting to the origin. Now add this fully searched leg to S and repeat until S contains all the legs. So the hider can't be anywhere, and the seeker must have found him.

Now we will show that Algorithm 3.4 guarantees capture of the hider in the shortest amount of time. To do this, we will compare the rate of clearing to the amount cleared. In Algorithm 3.4, each phase involves the seeker starting at the origin and moving up a leg and back to the origin. Let x denote the total distance traveled on that leg in a given phase so that the seeker moves $x/2$ up the leg and $x/2$ back down. The amount of area she clears is $x/2$. Unless the seeker goes to the end of the leg, the hider gains $wx/2$ on the searched leg. This gain could be caused by two possibilities, if there is some uncleared area on the leg after the seeker has gone in $x/2$, then this will expand by at least $wx/2$ as the seeker returns to the origin. Otherwise, the seeker forces the leg to have no uncleared area after this phase, in this case the uncleared area is expanding until the seeker has traveled $x/2$ in on the leg. Now there are some other legs which the hider

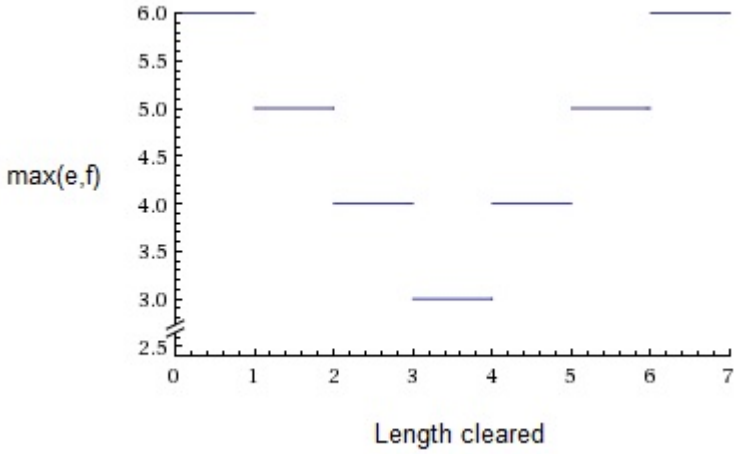

Figure 3.2 The graph of the upper bound on $\max(e, f)$.

will gain wx length in. In fact, the only legs for which this gain doesn't occur on are those completely cleared or completely uncleared. The center could be cleared or uncleared; this means a completely clear leg and a completely uncleared leg can't exist at the same time. Let e be the number of legs the hider can't be on at this point in time, and let f be the number of legs the hider could be anywhere on. So the minimum number of legs which the hider gains wx uncleared length on is $n - 1 - \max(e, f)$.

Combining the previous terms, the clearing rate of the hider is $1/2 - w/2 - (n - 1 - \max(e, f))w$. Now we want to maximize $\max(e, f)$ in order to make the clear rate as large as possible. We will look at the upper bound on e and f as a function of area cleared. In order for e legs to be completely clear, then the length cleared must be at least e. Also once we have cleared an length of c then the number of completely cleared legs is at most $n - c$. Since e and f must be integers, then the upper bound on e is $\lfloor c \rfloor$ and the upper bound on f is $\lfloor n - c \rfloor$. Figure 3.2 gives an upper bound on $\max(e, f)$, where c is the area cleared and $n = 7$.

Since the rate of clearing is $1/2 - w/2 - (n - 1 - \max(e, f))w$ its upper bound is simply a linear transformation of the upper bound of $\max(e, f)$. We will now return to the fact that this is not always the upper bound on clearing. If the seeker were to completely clear a leg, then the rate of clearing would be $1/2 - (n - 1 - \max(e, f))w$. This seems like it would be better and should be achieved as much as possible. Unfortunately, if the start of this clearing occurs and causes more legs to gain back length for

the hider then the rate of clearing does not decrease. If we are to fully clear a completely uncleared edge, then there are $n - \max(e, f)$ edges where the hider gains back uncleared area. So the rate of clearing is instead $1/2 - (n - \max(e, f))w$ which is worse than our upper bound. So an optimal algorithm would always completely clear a leg when the amount clear is exactly an integer when less than half the area is cleared, and when the amount cleared after the leg is checked will be an integer when more than half the area is cleared.

Algorithm 3.4 always has exactly an integer length cleared after each iteration. This can be seen by looking at the set of legs in S after each iteration. Also, depending on whether the algorithm is less than half way done or more than half way done determines when it completely clears a leg. These leg clearings line up with the line clearings which increase the rate of clearing listed above. Since at every point in the search process this algorithm obtains the upper bound on the rate of clearing then this algorithm must be optimal.

Interestingly, the upper bound on the clearing rate shows why the hider cannot be caught when $w \geq \frac{1}{n}$ when n is odd and $w \geq \frac{1}{n-1}$ when n is even. The clearing rate is the smallest when $\max(e, f)$ is as small as possible. This occurs when exactly half the area is cleared, this means $\max(e, f) = \lfloor \frac{n}{2} \rfloor$. The rate of clearing at this point is $1/2 - w/2 - (n - 1 - \lfloor \frac{n}{2} \rfloor)w$. When this maximum clearing rate is zero or negative at any point, then it is impossible to have any more length cleared. So it is impossible to guarantee capture of the hider when

$$1/2 - w/2 - (n - 1 - \lfloor \frac{n}{2} \rfloor)w \leq 0.$$

If we manipulate this to solve for w we get

$$\begin{aligned} 1/2 - w/2 - (n - 1 - \lfloor \frac{n}{2} \rfloor)w &\leq 0 \\ 1/2 - (\lceil \frac{n}{2} \rceil - 1/2)w &\leq 0 \\ (\lceil \frac{n}{2} \rceil - 1/2)w &\geq 1/2 \\ \frac{1}{2(\lceil \frac{n}{2} \rceil - 1/2)} &\leq w \\ \frac{1}{2\lceil \frac{n}{2} \rceil - 1} &\leq w. \end{aligned}$$

We will first compute this when n is odd. In this case we get

$$\frac{1}{2\frac{n+1}{2} - 1} \leq w$$

$$\frac{1}{n} \leq w.$$

Now computing this when n is even we get

$$\frac{1}{2\frac{n}{2} - 1} \leq w$$

$$\frac{1}{n-1} \leq w.$$

These are the bounds we gave on w originally. So not only does the above method show the optimality of this algorithm, it also provides proof that the seeker can't catch the hider for sure when the hider is moving fast enough.

Chapter 4

Searching on Infinite Stars

Now we will investigate the case when the legs of the stars extend infinitely. In the infinite case, all edges emanating from the center vertex extend infinitely.

Clearly, the seeker can never guarantee capture when $w \geq 1/n$ and n is odd, or when $w \geq \frac{1}{n-1}$ and n is even as the finite n-star is a subspace of the infinite n-star. Unfortunately, the last algorithm given for finite stars does not easily generalize to infinite stars. The first algorithm generalizes nicely and guarantees capture of the hider when $w < \frac{1}{2n-3}$. The oscillations which occur on the largest scale resemble those made in the original cow-path problem, to make sure the hider doesn't simply run away from the origin. We must maintain the oscillations from the finite case, so as to avoid the hider staying close to the origin.

Our algorithm starts by labeling the legs of the star from 1 to n. Now on the kth iteration, the seeker goes out r^k units on each leg except for the $k+1 \pmod n$ leg starting with the $k \pmod n$ leg, and visits the legs in decreasing order. She continues exploring these legs only going in $\frac{2(n-2)w}{1-w}$ of what she went in before on that same leg. When she converges to the origin, the seeker starts the next iteration. We now need to compute what r will guarantee capture of the hider. We know that the hider can't get close to the origin, since we check the area close to the origin at the end of each iteration. We need to verify that if the hider is moving away from the origin on any ray he will be caught.

The distance that the seeker travels on the first visit on legs on the kth iteration is $r^k 2(n-2)$. Then each iteration after that they only go in $\frac{(2n-4)w}{1-w}$

as far. So the total distance the seeker travels on the kth iteration is

$$\frac{r^k 2(n-2)}{1-\frac{(2n-4)w}{1-w}} = \frac{r^k(2n-4)(1-w)}{1-(2n-3)w}.$$

We need to show that no matter how far away from the origin the hider starts, he can't get away from the seeker. Let A be the distance from the origin the hider starts. The maximum distance from the origin that the hider can be at the start of the $k+1$st iteration is

$$(\frac{r^k(2n-4)(1-w)}{(1-(2n-3)w)(1-\frac{1}{r})} + r^{k+1})w + A.$$

There must be some r such that for all $A \geq 0$ there is some k such that

$$\begin{aligned}
(\frac{r^k(2n-4)(1-w)}{(1-(2n-3)w)(1-\frac{1}{r})} + r^{k+1})w + A &< r^{k+1} \\
(\frac{r^{k+1}(2n-4)(1-w)}{(1-(2n-3)w)(r-1)} + r^{k+1})w + A &< r^{k+1} \\
(\frac{(2n-4)(1-w)}{(1-(2n-3)w)(r-1)} + 1)wr^{k+1} + A &< r^{k+1}.
\end{aligned}$$

We also need to choose r so that $(\frac{(2n-4)(1-w)}{(1-(2n-3)w)(r-1)} + 1)w$ is less than one. Solving for r we get

$$\begin{aligned}
(\frac{(2n-4)(1-w)}{(1-(2n-3)w)(r-1)} + 1)w &< 1 \\
(\frac{(2n-4)(1-w)}{(1-(2n-3)w)} + r - 1)w &< r-1 \\
\frac{(2n-4)(1-w)w}{(1-(2n-3)w)} + rw - w &< r-1 \\
\frac{(2n-4)(1-w)w}{(1-(2n-3)w)} - w + 1 &< r(1-w) \\
\frac{(2n-4)w}{(1-(2n-3)w)} - 1 &< r.
\end{aligned}$$

We need that $r > 1$ though. So we simply choose $r > \max(1, \frac{(2n-4)w}{(1-(2n-3)w)} - 1)$. Since these are both finite we can choose an r which guarantees capture.

This algorithm can be made more efficient since the seeker doesn't need to check each leg as far out as given in the algorithm. We could also choose

r to minimize our competitive ratio. Unfortunately, computing the competitive ratio if the hider gets close to the origin instead of running away becomes complicated, and the worst-case scenario for capture in this case is still unknown.

Chapter 5

Searching on Trees with Finite Branching

Here we extend the algorithm for searching an infinite star to searching a finite tree. First the seeker chooses any node to be the origin. Then the seeker starts searching as if she was searching an infinite star. When the seeker reaches a node besides the origin and still has x distance left to travel out, then the seeker will start doing Algorithm 3.2 backwards for a total length of x and then do the same search algorithm forwards covering a length of x, where the leg the seeker came in on is the leg she doesn't check. The backwards and forwards finite star search at each node guarantees that the hider is not hiding close to any node. The ratio r now indicates the time the seeker spends on the subtree before returning to the origin, as she will traverse the same lengths multiple times. The ratio will have to now depend on the maximum number of nodes which occur on a path from the origin. Each node that the seeker traverses away from the origin decreases the length traveled away from the origin by a constant factor. Therefore, our computation above would be similar but with r^{k+1} multiplied by some constant on the right-hand side dependent on the maximum number of branchings on a path from the origin. Let x be this constant.

So we need to choose our r such that

$$(\frac{(2n-4)(1-w)}{(1-(2n-3)w)(r-1)}+1)w < x.$$

We can always choose r to be arbitrarily big, as long as $w < x$. So we actually cannot have infinite branching and still maintain the same feasible w we had before. This does allow us to catch a hider on a tree assuming that the hider is moving slowly enough.

Chapter 6

Future Work

We have found an algorithm which guarantees capture on a finite star, when the hider's maximum speed is sufficiently small. Not only that, but we have shown that the seeker cannot guarantee to catch the hider in less time. We've also shown that for faster maximum speeds of the hider the seeker can no longer guarantee capture of the hider. We have also found some algorithms for searching on infinite stars and trees which generalize the algorithms given.

This still leaves plenty of questions open. The strategy given for an omnipotent hider only works when $w \geq 1/2$. A possible extension is to find a strategy for the hider for smaller w. It seems as though the hider may be able to base their strategy off of the uncleared area. Another area of investigation is to find the optimal algorithm for infinite stars, and compute the competitive ratio in this case. Another extension is to simply find an optimal algorithm for searching finite trees. Lastly, all the algorithms given in this paper have been completely deterministic, but sometimes bounds can be greatly improved via randomized algorithms. This leaves the area on randomized algorithms on these search spaces completely open.

Bibliography

Alpern, Steve, and Shmuel Gal. 2002. *The Theory of Search Games and Rendezvous*, *International Series in Operations Research and Management Science*, vol. 55. Kluwer Academic Publishers.

Kao, Ming-Yang, John H. Reif, and Stephen R. Tate. 1996. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Information and Computation* 131:63–80.

Papadimitriou, Christos, and Mihalis Yannakakis. 1991. Shortest paths without a map. *Theoretical Computer Science* 84:127–150.

von Stengal, Bernhard, and Ralph Werchner. 1997. Complexity of searching an immobile hider in a graph. *Discrete Applied Mathematics* 78:235–249.