Search-and-Fetch with One Robot on a Disk ⋆

(Track: Wireless & Geometry)

Konstantinos Georgiou[1], George Karakostas[2], and Evangelos Kranakis[3]

[1] Ryerson University, Dept. of Mathematics, Toronto, Ontario, Canada.
[2] McMaster University, Dept. of Computing and Software, Hamilton, Ontario, Canada.
[3] Carleton University, School of Computer Science, Ottawa, Ontario, Canada.

Abstract. A robot is located at a point in the plane. A treasure and an exit, both stationary, are located at unknown (to the robot) positions both at distance one from the robot. Starting from its initial position, the robot aims to fetch the treasure to the exit. At any time the robot can move anywhere on the disk with constant speed. The robot detects an interesting point (treasure or exit) only if it passes over the exact location of that point. Given that an adversary controls the locations of both the treasure and the exit on the perimeter, we are interested in designing algorithms that minimize the treasure-evacuation time, i.e. the time it takes for the treasure to be discovered and brought to the exit by the robot.
In this paper we differentiate how the robot's knowledge of the distance between the two interesting points affects the overall evacuation time. We demonstrate the difference between knowing the exact value of that distance versus knowing only a lower bound and provide search algorithms for both cases. In the former case we give an algorithm which is off from the optimal algorithm (that does not know the locations of the treasure and the exit) by no more than $\frac{4\sqrt{2}+3\pi+2}{6\sqrt{2}+2\pi+2} \leq 1.019$ multiplicatively, or $\frac{\pi}{2} - \sqrt{2} \leq 0.157$ additively. In the latter case we provide an algorithm which is shown to be optimal.
Key words and phrases. Disk, Exit, Robot, Search and Fetch, Treasure.

1 Introduction

Search is concerned with finding an object under various conditions within a search space. In the context of computational problems this usually becomes more challenging especially when the environment is unknown to the searcher (see [1,3,24]) and efficient algorithms with respect to search time are sought. For example, in robotics exploration may be taking place within a given geometric domain by a group of autonomous but communicating robots and the ultimate goal is to design an algorithm so as to accomplish the requirements of the search (usually locating a target of unknown a priori position) while at the same time obeying the computational and geographical constraints. Further, the task must be accomplished in the minimum possible amount of time [8].

There is extensive research and several models have been proposed and investigated in the mathematical and theoretical computer science literature with

⋆ Research supported in part by NSERC Discovery grants.

particular emphasis on probabilistic search [24], game theoretic applications [3], cops and robbers [9], classical pursuit and evasion [23], search problems as related to group testing [1], searching a graph [22]. A survey of related search and pursuit evasion problems can be found in [11], whereby pursuers want to capture evaders trying to avoid capture. Examples include *Cops and Robbers* (where the cops try to capture the robbers by moving along the vertices of a graph), *Lion and Man* (a geometric version of cops and robbers where a lion is to capture a man in either continuous or discrete time), etc. Searching for a stationary point target has some similarities with the lost at sea problem, [17,18], the cow-path problem [6,7], and with the plane searching problem [5].

In this paper, we study a new problem which involves a robot *searching* for a treasure and *fetching* it to an exit. Both treasure and exit are at distance 1 from the starting position of the robot (i.e., located on the perimeter of a unit radius circle) at locations unknown to the robot. The robot can move with maximum speed 1, starts at the centre of a circle and continues by moving to the perimeter. The adversary has control over the locations of both the exit and the treasure. Goal is to provide algorithms that minimize the "search time" for the robot to find the treasure and bring it to the exit. Surprisingly, finding an optimal algorithm turns out to be a rather difficult problem even when the robot has some knowledge on the arc-distance between exit and treasure.

There are several problems in the scientific literature relating to evacuation, although of very different nature than our problem. For grid polygons, evacuation has been studied in [16] from the perspective of constructing centralized evacuation plans, resulting in the fastest possible evacuation time from the rectilinear environment. Our problem has similarities to the well-known evacuation problem on an infinite line (see [4] and the more recent [10]) in that the search is for an unknown target; an important difference is that, in the basic optimal zigzag algorithm presented in [4], the search is on an infinite line which limits the possibilities for the adversary. Additional research and variants on this problem can be found in [15] (on searching with turn costs), [21] (randomized algorithm for the cow-path problem), and [20] (hybrid algorithms).

Our model is relevant to the recent works [12,13,14] investigating algorithms in the wireless and non-wireless (or face-to-face) communication models for the evacuation of a team of robots. Note that in this case, the "search domain" is the same and the evacuation problem without a treasure for a single robot is trivial. Thus, in addition to searching for the two stationary objects (namely treasure and exit) at unknown locations in the perimeter of a cycle we are also interested in fetching the treasure to the exit. As such, our search-and-fetch type problem is of much different nature than the series of evacuation-type problems above, and in fact solutions to our problem require a novel approach.

Our optimization problem models real-life situations that may arise in surveillance, emergency response, and search-and-rescue operations, e.g. by aerial drones or other unmanned vehicles. Indeed, consider a rescue-robot that receives a distress signal indicating that a victim is at an unknown location but at known distance from a safe shelter. What is the optimal trajectory of the robot that can locate the victim and fetch it to the shelter? Similar problems are well studied in the robotics community since the 90's, e.g. see [19]. A search-and-fetch problem similar to ours was introduced by Alpern in [2], where the underlying domain was discrete and the approach/analysis resembled that of standard search-type problems [3]. In contrast, the focus of the current work is to demonstrate how some knowledge of the input may affect optimality in designing online solutions when there is only one rescue-robot available.

2 Preliminaries, Notation, and Results of the Paper

We begin with presenting the precise definitions of the treasure evacuation problem and some basic notation, concepts and necessary definitions.

A treasure and an exit are located at unknown positions on the perimeter of a unit-disk and at arc distance α (in what follows all distances will be arc-distances, unless specified otherwise). A robot starts from the center of the disk, and can move anywhere on the disk at constant speed 1. The robot detects the treasure or the exit only if its trajectory passes over that point on the disk. Once detected, the treasure can be carried by the robot at the same speed. Our goal is to design algorithms that minimize the evacuation time, i.e. the time it takes from the moment the robot starts moving, till the treasure is detected and brought to the exit by the robot. Sometimes we refer to the task of bringing the treasure to the exit as *treasure-evacuation*. We also use the abbreviations T, E for the treasure and the exit, respectively. For convenience, in the sequel we will refer to the locations of the exit and the treasure as *interesting* points. For an interesting point I on the perimeter of the disk, we also write $I = E$ ($I = T$) to indicate that the exit (treasure) lies at point I.

We focus on the following two variants of treasure-evacuation reflecting the knowledge of the robot with respect to its environment.

Definition 1. *In* ***1-TE***$_{=}$*, one robot attempts treasure-evacuation knowing that the distance between* T, E *is* exactly α*. In* ***1-TE***$_{\geq}$*, one robot attempts treasure-evacuation knowing that the distance between* T, E *is* at least α*.*

2.1 Final Steps of Treasure-Evacuation

The exploration part of any evacuation algorithm concludes with the discovery of an interesting point (i.e., treasure or exit). This leads us to define the concepts of *double* and *triple* move that will prove useful in the subsequent analysis for

the case when the robot knows that the exit and the treasure are at distance exactly α. If a robot encounters an interesting point at I then the other interesting

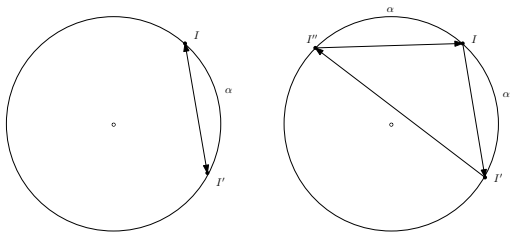

Fig. 1. The double move (left) and triple move (right) of a robot occur when the first interesting point to be encountered by the robot is an exit.

point is either at point I' or I'' (Figure 1, right), both at arc distance α from I.

If the robot has already explored one of these points (say I'') without finding an interesting point there, then the situation reduces to the left of Figure 1. If this is the robot that will eventually evacuate the treasure, then the worst case is to find the exit in I and the treasure in I'; in this case, the robot will perform a *double move*, going to I', pick up the treasure, and return to I, spending time equal to twice the chord distance between I and I', i.e. $4\sin(\alpha/2)$. On the other hand, if both I', I'' are still unexplored, the worst case for the algorithm would be to perform a *triple move*, i.e., find the exit at I, then visit I' without finding anything there, then visit I'' and pick up the treasure, and return to I, always moving along the shortest underlying chords, inducing time $4\sin(\alpha/2)+2\sin(\alpha)$. Once an interesting point is found, we will refer to the above process as the evacuation step. Clearly, this will involve either a double or a triple move, depending on whether sufficiently many points on the disk have already been explored.

2.2 Outline and results of the paper

We study treasure evacuation for a single robot. We contrast how knowledge affects the evacuation time by considering two models: in the first the robot knows only a lower bound on the actual arc-distance, while in the second the robot knows the distance between treasure and exit. When a lower bound α on the actual arc-distance l is known to the robot (see Section 3) then we give an optimal treasure evacuation algorithm which takes time $1+2\pi-\alpha+2\sin(\alpha/2)+2\sin(l/2)$. When the robot knows that α is not just a lower bound, but the *actual* arc-distance between treasure and exit (i.e., $l = \alpha$) (see Section 4), then we propose the *arc-partition* algorithm which makes the robot alternate between exploring and hopping arcs and use continuous optimization techniques in order to show that it is nearly optimal. More specifically, our upper and lower bounds

are off multiplicatively by no more than $\frac{4\sqrt{2}+3\pi+2}{6\sqrt{2}+2\pi+2} \leq 1.019$, when $\alpha = \pi/2$, while for $\alpha \to 0$ or $\alpha \to \pi$ our algorithm is nearly optimal. In Section 5 we conclude and suggest several extensions and open problems.

3 Knowledge of Lower Bound on Arc Distance: Problem 1-TE$_{\geq}$

In this section we prove tight upper and lower bounds for treasure evacuation when only a lower bound, say α, on the arc distance between exit and treasure is known to the robot. Note that for problem 1-TE$_{\geq}$ considered here the special case $\alpha = 0$ means that the robot does not know anything about the arc distance between treasure and exit.

Our "Arc Avoidance" Evacuation Algorithm is given below. As implied by its name, the algorithm merely avoids exploration of an arc of length α following the encounter of the first interesting point.

Algorithm 1 Arc Avoidance Evacuation Algorithm

Step 1. Starting at the center of the circle, move to the perimeter and start exploring Clock-Wise (CW) along the perimeter until the first interesting point A is found.

Step 2. Move CW along chord AB of length $2\sin(\alpha/2)$ (see Figure 2).

Step 3. Continue exploring from point B on the perimeter until the second interesting point is found.

Step 4. Evacuate the treasure using either a double or a triple move.

Theorem 1. *When the robot is given a lower bound α on the actual arc distance l between exit and treasure, the evacuation time of Algorithm 1 is at most $1 + 2\pi - \alpha + 2\sin(\alpha/2) + 2\sin(l/2)$ in the worst-case. Further, Algorithm 1 is worst-case optimal, i.e. no algorithm can attain a better time in the worst case for a robot to evacuate the treasure.*

Proof. (Theorem 1) We prove the upper and lower bounds separately.

3.1 Upper Bound

Let x be the time it takes until the robot encounters at A the first interesting point on the perimeter of the disk. Consider the two cases depicted in Figure 2.

Case 1: $x > \alpha$. This is depicted in Figure 2 (left). The robot makes the move $A \to B$ along the chord. Then it explores again the perimeter. Let y be the time until it finds the second interesting point at C. It is clear that the worst total cost (i.e., $A = E, C = T$) is

$$1 + x + 2\sin(\alpha/2) + y + 2\sin(l/2), \tag{1}$$

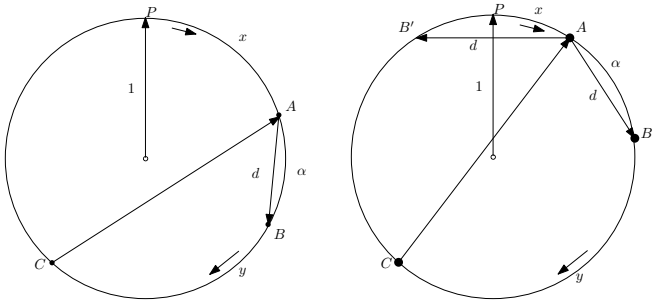

Fig. 2. The two cases of the evacuation Algorithm 1.

where $2\sin(l/2)$ is the length of the chord CA connecting the treasure to the exit. Further, observe that $y \leq 2\pi - \alpha - x$, which implies that $x + y \leq 2\pi - \alpha$. Also, for this case we have that $x \geq \alpha$ and, since $y \leq 2\pi - \alpha - x$, the bound follows.

Case 2: $x \leq \alpha$. This is depicted in Figure 2 (right). Let B, B' be the two points at arc distance α from A, and y be the time it takes for the robot to find the second interesting point at C. As before, the evacuation cost is given by Formula (1). Observe that in this case the robot does not need to traverse the arc $\overset{\frown}{B'AB}$ (by assumption the other interesting point must be at arc distance at least α) and therefore $y \leq 2\pi - 2\alpha$. Also since $x \leq \alpha$ we also have $x + y \leq 2\pi - \alpha$, and the bound follows.

3.2 Lower Bound

First we prove the following lemma which will be used in the proof of the lower bound below.

Lemma 1. *If a robot during its exploration of the perimeter has covered length less than* $2\pi - \alpha$ *then there is a chord of length exactly* $2\sin(\alpha/2)$ *none of whose endpoints has been explored by the robot.*

Proof. (Lemma 1) Let us define $d := 2\sin(\alpha/2)$. Assume on the contrary no such chord exists. It follows that for any unexplored point A if we draw two chords AA_1 and AA_2 each of length d then each point in the arc A_1A_2 must have been explored by the robot (see Figure 3).

The same observation holds for any unexplored point in either of the arcs AA_1 and AA_2. If we consider the leftmost and righmost unexplored points on the circle on either side of A, say B and C, then the arc BA_1A_2C is fully explored and the distance between B and C must be at least d (if $|BC| < d$

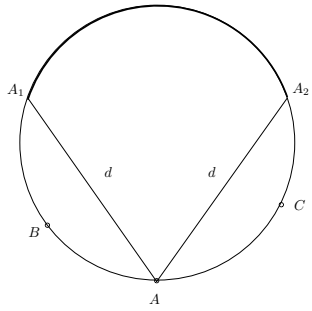

Fig. 3. Proving that there is a chord of length at least d with two unexplored endpoints.

then the explored portion would exceed $2\pi - \alpha$, a contradiction, since the robot does not have enough time to explore the arc BA_1A_2C). This proves the lemma. □

We now return to the proof of the main theorem. Consider any algorithm $\mathcal{A}$ solving the evacuation problem for a single robot, an exit and a treasure. Run the algorithm starting at the centre of the circle. It takes at least one time unit for the robot to reach the perimeter. The robot starts exploring the perimeter after this time. Run algorithm $\mathcal{A}$ for additional $2\pi - \alpha - \epsilon$ time. During this additional time, the robot can explore a total length of at most $2\pi - \alpha - \epsilon$ of the perimeter.

Take a pair of points delimiting a chord with the properties specified in Lemma 1. Such a chord has length exactly $d = 2\sin(\alpha/2)$ and has both endpoints unexplored. Clearly, the robot has not yet found neither the exit nor the treasure. Following the algorithm $\mathcal{A}$ the robot will visit one of its endpoints first; the adversary can place the exit at this first endpoint and the treasure in the other. Therefore it will take additional time at least $2\sin(l/2)$ to evacuate the treasure, and the total evacuation time will be at least $1 + 2\pi - \alpha - \epsilon + 2\sin(\alpha/2) + 2\sin(l/2)$, for any $\epsilon > 0$. This completes the proof of Theorem 1. □

4 Exact Knowledge of Arc-Distance: Problem 1-TE$_=$

In this section, we separate our analysis into an upper bound accompanied by a search algorithm and a lower bound using continuous optimization techniques.

4.1 Upper Bound

When a robot knows the exact arc-distance of the interesting points, there is a naive evacuation protocol, which we call the *Sweeping Algorithm*: go to an arbitrary point on the perimeter of the circle, and start traversing clockwise till an interesting point is found. Then either perform a double or a triple move to

finalize the evacuation. It is not hard to see that in the worst case, this naive algorithm has cost $1 + 2\pi - \alpha + 4\sin(\alpha/2)$. Our goal in this section is to improve upon this naive approach. To explain the main evacuation algorithm we first provide a definition.

Definition 2 (Alternating Arcs Partition). *An alternating arcs partition is a partition $\mathcal{P}$ of the perimeter of the cycle into $2n+1$, $n \geq 0$ consecutive and pairwise non-intersecting arcs $a_1, b_1, a_2, b_2, \ldots, a_n, b_n, a_{n+1}$, such that $b_i \leq \alpha$, $\forall i$. If in addition we have that $\alpha \leq a_i$, we call the partition α-greedy.*

Our Algorithm 2 first picks an alternating arc partitioning $\mathcal{P}$, before it follows through with its steps. In case the partition contains only a_1 (i.e., $n = 0$), the algorithm performs no "jumps" along chords until the first interesting point is encountered, i.e. it is exactly the Sweeping Algorithm.

Algorithm 2 Alternating Arcs, on input partition $\mathcal{P}$

Step 1. Walk to the perimeter, say point P;
Step 2. Keep moving in CW direction *arc-sweeping* each a_i, and *cord-jumping* along the chord of each b_i, until the first interesting point is found.
Step 3. Evacuate the treasure using either a double or a triple move.

Algorithm 2 succeeds in evacuating the treasure when run on input some α-greedy partition. Correctness is an immediate corollary of the definition of the partition (that depends on the value of α). Note that the *chord-jumps* correspond to arcs of size at most α, hence two interesting points cannot fit in any arc b_i. Also, since the *sweeping-arcs* a_i have length at least α, two interesting points cannot be in two (consecutive) arcs b_i, b_{i+1}. It follows that, eventually, a robot following Algorithm 2 will locate an interesting point and will evacuate with either a double or a triple move.

It remains to define the partition $\mathcal{P}$ used by Algorithm 2. The intuition behind our partition is based on the following greedy rule: perform as many alternating arc-chord moves as possible so that the worst configuration of interesting points which concludes evacuation with a double-move has total time no more than the worst configuration of interesting points which concludes evacuation with a triple-move.

Definition 3 (Greedy Partition $\mathcal{P}_\alpha$). *For every $\alpha > 0$, we set*

$$\kappa_\alpha := \left\lfloor \frac{2\pi - 3\alpha - 2\sin(\alpha)}{2\alpha} \right\rfloor, \quad \textit{slack}_\alpha := 2\pi - (2\kappa_\alpha + 3)\alpha - 2\sin(\alpha)$$

and

$$\gamma_\alpha := \begin{cases} \alpha & ,\textit{if } \textit{slack}_\alpha > \alpha \\ \textit{slack}_\alpha & ,\textit{otherwise} \end{cases}, \quad \chi_\alpha := \begin{cases} \textit{slack}_\alpha - \alpha & ,\textit{if } \textit{slack}_\alpha > \alpha \\ 0 & ,\textit{otherwise} \end{cases}$$

Then, we define

$$\mathcal{P}_\alpha := \begin{cases} a_1, b_1, \ldots, a_{\kappa_\alpha}, b_{\kappa_\alpha}, a_{\kappa_\alpha+1}, b_{\kappa_\alpha+1}, a_{\kappa_\alpha+2} & ,\textit{if } \kappa_\alpha \geq 0 \\ 2\pi & ,\textit{otherwise} \end{cases}$$

where $a_i = b_i = \alpha$*, for* $i = 1 \ldots \kappa_\alpha$*,* $a_{\kappa_\alpha+1} = \alpha$*,* $b_{\kappa_\alpha+1} = \gamma_\alpha$*, and* $a_{\kappa_\alpha+2} = 2\alpha + 2\sin(\alpha) + \chi_\alpha$*.*

It is clear that if $\kappa_\alpha < 0$, then the above partition gives rise to the sweeping algorithm, which happens after the root $\alpha_0 \approx 1.43396$ of the equation $2\pi = 3\alpha + 2\sin(\alpha)$. It is also clear that $\mathcal{P}_\alpha$ is an α-greedy alternating arcs partition.

Next we show that the worst configuration for Algorithm 2 that uses partition $\mathcal{P}_\alpha$ is indeed a double-move.

Lemma 2. *The worst case configuration for Algorithm 2 that uses partition* $\mathcal{P}_\alpha$ *makes the robot perform a double-move. An upper bound for the cost can be computed by placing the treasure arbitrarily close to the starting point* P *(from the opposite direction than the one that the robot starts sweeping) and the exit at CCW-arc-distance* α *from it.*

Proof. The statement is true for the sweeping algorithm, since the worst-case double-move is no less costly than the worst triple-move. So we may focus on the case $\text{slack}_\alpha > 0$, in which we do have at least one chord-move.

We observe that after the last chord-move, the robot is at some point A at distance $2\alpha + 2\sin(\alpha) + \chi_\alpha$ away from returning to the original point P. If it is forced to do any double or triple move after point A, that would make the total cost at least as costly as in any other double or triple move (respectively) configuration before that.

Next recall that the cost of the double-move alone is $4\sin(\alpha/2)$, while that of a triple-move (alone) is $4\sin(\alpha/2) + 2\sin(\alpha)$. The worst-case scenario ending with a double-move would be to have the treasure close to P and the exit at distance α from P in a Counter-CW (CCW) direction; by construction, the robot will have to sweep an additional arc of length $\alpha + 2\sin(\alpha) + \chi_\alpha - \epsilon$ before encountering an interesting point, i.e. before the double-move starts. In contrast, the worst-case triple-move scenario would be possible only if the robot discovered an interesting point in distance $\alpha - \epsilon$ after point A. The fact that χ_α is always non negative shows the claim. This proves Lemma 2. □

Now we can give an upper bound to the performance of our algorithm.

Theorem 2. *For every $\alpha > 0$, the performance of Algorithm 2 that uses partition $\mathcal{P}_\alpha$ is at most*

$$\begin{cases} 2\pi - (\kappa_\alpha + 2)\alpha + 2(\kappa_\alpha + 3)\sin(\alpha/2)\,, & \text{if } \pi > (\kappa_\alpha + 2)\alpha + \sin\alpha \\ (\kappa_\alpha + 2)\alpha + 2(\kappa_\alpha + 2)\sin(\alpha/2) + 2\sin\left(\frac{2\kappa_\alpha+3}{2}\alpha + \sin\alpha\right) + 2\sin\alpha, & \text{o.w.} \end{cases}$$

where $\kappa_\alpha = \left\lfloor \frac{2\pi - 3\alpha - 2\sin(\alpha)}{2\alpha} \right\rfloor$.

Proof. By Lemma 2, the worst configuration will place the treasure close to P and the exit at distance α from P in a CCW direction. Hence, using the α-greedy partition $\mathcal{P}_\alpha$, the cost of Algorithm 2 is $1 + \sum_{i=1}^{\kappa_\alpha+1} a_i + 2\sum_{i=1}^{\kappa_\alpha+1} \sin(b_i/2) + (a_{\kappa_\alpha+2} - \alpha) + 4\sin(\alpha/2)$. Using Definition 3, we can expand the above formula with respect to whether $\kappa_\alpha \geq 0$ and $\text{slack}_\alpha > \alpha$ or not. Observing that $\kappa_\alpha \geq -1$ for all α and by simplifying the resulting formula yields the promised expression. Note that when $\kappa_\alpha = -1$, the above formula indeed induces the cost of the sweeping algorithm. □

4.2 Lower Bound

Next we prove our main lower bound for the problem 1-TE$_=$. In general, arguments for lower bounds are based on an adversary detecting the input configuration that maximizes the worst-case time of an optimal algorithm (for that configuration). We are not able to analyze such an all-powerful adversary (which would give us the best (i.e., highest) lower bound). Instead, we analyze an adversary that is restricted as follows to place the exit and the treasure only when the algorithm has left only one completely unexplored pair of points A, B at distance α, and only at the points A and B, therefore obtaining a weaker lower bound.

Theorem 3. *Any algorithm that solves problem 1-TE$_=$ must run for time at least*

$$1 + \pi + \min\left\{ \begin{array}{l} 4\sin\frac{\alpha}{2} + 2\left(\left\lceil \frac{\pi}{\alpha} \right\rceil - 1\right)\sin\frac{\pi-\alpha}{2\left(\left\lceil \frac{\pi}{\alpha} \right\rceil - 1\right)}, \\ \pi - \alpha\left\lfloor \frac{\pi}{\alpha} \right\rfloor + 2\left(\left\lfloor \frac{\pi}{\alpha} \right\rfloor + 1\right)\sin\frac{\alpha}{2} \end{array} \right\}$$

Proof. We restrict the adversary to place the exit and the treasure only when the algorithm has left only one completely unexplored pair of points A, B at distance α, and only at the points A and B. Obviously, with such a restricted adversary and because of the triangle inequality, the optimal algorithm will follow an alternating arcs partition in a single direction (say CW), starting at A,

ending its exploration phase at B, and leaving the arc AB of length α completely unexplored; at this point, the adversary will have placed the treasure at A and the exit at B, forcing a double-move.

Therefore, we have that $1 + \sum_{i=1}^{l} a_i + \sum_{i=1}^{l-1} 2\sin(b_i/2) + 4\sin(\alpha/2)$ is a lower bound on the evacuation time of an algorithm that chooses an alternating arcs partition, leaving a single unexplored segment b_l, and then performs a double move. Since $1 + 4\sin(\alpha/2)$ doesn't depend on the lengths of a_i, b_i, we get a lower bound for the optimal algorithm by minimizing $\sum_{i=1}^{l} a_i + \sum_{i=1}^{l-1} 2\sin(b_i/2)$, given the constraints we have imposed so far. This leads to the following family of optimization problems (one for each integer $l \geq 1$):

$$\textbf{minimize} \quad \sum_{i=1}^{l} a_i + \sum_{i=1}^{l-1} 2\sin(b_i/2) \tag{MP}$$

$$\textbf{subject to} \quad \sum_{i=1}^{l} a_i + \sum_{i=1}^{l} b_i = 2\pi \tag{2}$$

$$b_i \leq \alpha \text{ for } i = 1, 2, \ldots, l \tag{3}$$

$$a_i, b_i \geq 0 \text{ for } i = 1, 2, \ldots, l \tag{4}$$

which is equivalent to

$$\textbf{minimize} \quad -\sum_{i=1}^{l} b_i + 2\sum_{i=1}^{l-1} \sin(b_i/2) \tag{MP'}$$

$$\textbf{subject to} \quad b_i \leq \alpha \text{ for } i = 1, 2, \ldots, l \tag{5}$$

$$b_i \geq 0 \text{ for } i = 1, 2, \ldots, l \tag{6}$$

The overall lower bound we will calculate is the minimum amongst the candidate lower bounds calculated for each value of $l \geq 1$. Note that for $l = 1$, the bound corresponds to the case of a sweeping algorithm with $a_1 = 2\pi - b_1$. Given our adversary assumption, we have $b_1 = \alpha$. In this case the candidate lower bound for the running time is

$$LA = 1 + 4\sin(\alpha/2) + 2\pi - \alpha. \tag{7}$$

We estimate now the candidate lower bounds $LB(l)$ for $l \geq 2$ using (MP'). For brevity reasons, we set $x := \sum_{i=1}^{l} a_i$. Constraint (2) implies that $x \leq 2\pi - b_l$. Also, we have $\pi \leq x$, because of the following claim:

Claim. $\sum_{i=1}^{l} a_i \geq \sum_{i=1}^{l} b_i$.

Proof. Moving clockwise, and starting from b_1, we can map each point of $b_1, b_2, \ldots$ to a unique explored point of some a_i at distance α clockwise, since there is

only one pair of unexplored points in distance α when the first interesting point is found (at one of these two points). Therefore, the total length of the b_i's is mapped one-to-one to the a_i's, and the claim follows. □

For the special case $x = 2\pi - b_l$, we have that $b_i = 0, i = 1, \ldots, l-1$ (from (2)), and the bound becomes the bound in (7) (since all the a_i's form a single contiguous explored segment and then $b_l = \alpha$, according to our adversary assumption). Therefore we can assume that

$$\pi \leq x < 2\pi - b_l. \tag{8}$$

We note that

$$\frac{2\pi - b_l - x}{l-1} \leq \alpha \Leftrightarrow l \geq 1 + \frac{2\pi - x - b_l}{\alpha}, \tag{9}$$

because, otherwise, $2\pi > (l-1)\alpha + b_l + x \geq \sum_{i=1}^{l} a_i + \sum_{i=1}^{l} b_i = 2\pi$, a contradiction. We distinguish two cases:

Case 1: $\frac{2\pi - x - b_l}{l-1} < \alpha$

Suppose that an optimal solution of (MP') has a pair of optimal values $b_i < \frac{2\pi - b_l - x}{l-1}$ and $b_j > \frac{2\pi - b_l - x}{l-1}$ (note that if there is such a b_i, there must be such a b_j, and vice versa, due to (2)). Then there is $\varepsilon > 0$ such that $b_i + \varepsilon < \frac{2\pi - b_l - x}{l-1}$ and $b_j - \varepsilon > \frac{2\pi - b_l - x}{l-1}$. Notice that

$$\sin\left(\frac{b_i + \varepsilon}{2}\right) + \sin\left(\frac{b_j - \varepsilon}{2}\right) < \sin\left(\frac{b_i}{2}\right) + \sin\left(\frac{b_j}{2}\right),$$

since the LHS is a monotonically decreasing function of ε, a contradiction since by setting b_i, b_j to $b_i + \varepsilon, b_j - \varepsilon$ we get a better feasible solution of (MP') than the optimal. Hence

$$b_i = \frac{2\pi - b_l - x}{l-1},\ i = 1, \ldots, l-1 \tag{10}$$

and (MP') is equivalent of solving the following minimization problem

$$\min_{\pi \leq x < 2\pi - b_l} x + 2(l-1)\sin\frac{2\pi - b_l - x}{2(l-1)}.$$

If we set $y := \frac{2\pi - b_l - x}{2(l-1)}$, the latter is equivalent to solving

$$\min_{0 < y \leq \frac{\pi - b_l}{2(l-1)}} -b_l - 2(l-1)y + 2(l-1)\sin y.$$

For any value of b_l, the optimal solution has to minimize

$$\min_{0 < y \leq \frac{\pi - b_l}{2(l-1)}} \sin y - y$$

and, therefore, $y = \frac{\pi - b_l}{2(l-1)}$ in the optimal solution. This also implies that $x = \pi$. Hence, the objective function becomes a function of b_l:

$$\min_{0 \le b_l \le \alpha} 2(l-1) \sin \frac{\pi - b_l}{2(l-1)}$$

Since $\frac{\pi - b_l}{2(l-1)} \le \frac{\pi}{2}$, the minimum is achieved for $b_l = \alpha$. Therefore $b_i = \frac{\pi - \alpha}{l-1}$, $i = 1, \ldots, l-1$ (note that $\frac{\pi-\alpha}{l-1} \le \alpha$ because of (9)). In this case, the lower bound candidate is

$$LB(l) = 1 + 4 \sin \frac{\alpha}{2} + \pi + 2(l-1) \sin \frac{\pi - \alpha}{2(l-1)}. \tag{11}$$

(11) is increasing in l, and, therefore, the smallest $LB(l)$ is (because of (9))

$$LB\left(\left\lceil \frac{\pi}{\alpha} \right\rceil\right) = 1 + 4 \sin \frac{\alpha}{2} + \pi + 2\left(\left\lceil \frac{\pi}{\alpha} \right\rceil - 1\right) \sin \frac{\pi - \alpha}{2(\lceil \frac{\pi}{\alpha} \rceil - 1)} \tag{12}$$

if $\frac{\pi}{\alpha} \notin \mathbb{N}$, and

$$LB\left(\frac{\pi}{\alpha} + 1\right) = 1 + 4 \sin \frac{\alpha}{2} + \pi + \frac{2\pi}{\alpha} \sin \frac{\alpha(\pi - \alpha)}{2\pi} \tag{13}$$

if $\frac{\pi}{\alpha} \in \mathbb{N}$.

Case 2: $\frac{2\pi - x - b_l}{l-1} = \alpha \Leftrightarrow x = 2\pi - b_l - \alpha(l-1)$

In this case, (5) implies that $b_i = \alpha, i = 1, \ldots, l-1$. Also, (8) implies that $b_l \le \pi - \alpha(l-1)$. Since $b_l \ge 0$, this implies that $l \le 1 + \frac{\pi}{\alpha}$. The candidate lower bounds (parameterized by l) are

$$LC(l) = 1 + 4 \sin \frac{\alpha}{2} + 2\pi - b_l - \alpha(l-1) + 2(l-1) \sin \frac{\alpha}{2}.$$

Then, problem (MP') becomes

$$\min_{0 \le b_l \le \min\{\pi - \alpha(l-1), \alpha\}} -b_l + \sin \frac{b_l}{2}$$

which is optimized when $b_l = \min\{\pi - \alpha(l-1), \alpha\}$. If $\pi - \alpha(l-1) < \alpha$, then the minimum candidate is

$$LC\left(\left\lceil \frac{\pi}{\alpha} \right\rceil\right) = 1 + \pi + 2\left(\left\lceil \frac{\pi}{\alpha} \right\rceil + 1\right) \sin \frac{\alpha}{2} \tag{14}$$

if $\frac{\pi}{\alpha} \notin \mathbb{N}$, and

$$LC\left(\frac{\pi}{\alpha} + 1\right) = 1 + \pi + \left(\frac{2\pi}{\alpha} + 4\right) \sin \frac{\alpha}{2} \tag{15}$$

if $\frac{\pi}{\alpha} \in \mathbb{N}$. Since (15) is always greater or equal to the bound in (13), we will consider only the latter. If $\pi - \alpha(l-1) \geq \alpha$, then we have that $2 \leq l \leq \frac{\pi}{\alpha}$ and the minimum candidate is

$$LC\left(\left\lfloor \frac{\pi}{\alpha} \right\rfloor\right) = 1 + 2\pi - \alpha \left\lfloor \frac{\pi}{\alpha} \right\rfloor + 2\left(\left\lfloor \frac{\pi}{\alpha} \right\rfloor + 1\right) \sin\frac{\alpha}{2} \tag{16}$$

which holds only for $\alpha \leq \pi/2$. But (16) is lower than (7) for $\alpha \leq \pi/2$, and coincides with it for $\pi/2 \leq \alpha \leq \pi$. It is also lower than (15) for $\frac{\pi}{\alpha} \in \mathbb{N}$. Hence we will not be considering (7) at all.

The lower bound (for $\frac{\pi}{\alpha} \notin \mathbb{N}$) will be $\min\{LB(\lceil \frac{\pi}{\alpha} \rceil), LC(\lceil \frac{\pi}{\alpha} \rceil), LC(\lfloor \frac{\pi}{\alpha} \rfloor)\}$. But in this case, note that $LB(\lceil \frac{\pi}{\alpha} \rceil) \leq LC(\lceil \frac{\pi}{\alpha} \rceil)\ \forall \alpha$, so the lower bound will be $\min\{LB(\lceil \frac{\pi}{\alpha} \rceil), LC(\lfloor \frac{\pi}{\alpha} \rfloor)\}$. □

5 Qualitative Discussion About Our Results / Conclusions

We introduced a new optimization problem on *searching and fetching* with one robot from a unit disk, where there is limited information about the input. We studied two variants reflecting knowledge the robot has about its environment and contrasted how robot knowledge and capabilities affect the search time.

The goal of the current work was to provide nearly optimal algorithms with respect to worst case analysis. Indeed, in Section 3 we provided an optimal algorithm for problem 1-TE$_{\geq}$. Problem 1-TE$_{=}$was studied in Sections 4. The proposed algorithm is multiplicatively off from the optimal by at most the ratio between the expression derived in Theorem 2 (upper bound) over the expression derived in Theorem 3 (lower bound). It is not difficult to see that this value tends to 1 when α tends to either 0 or π, meaning that our algorithm is nearly optimal. Also, some tedious calculations show that this ratio is at most $\frac{4\sqrt{2}+3\pi+2}{6\sqrt{2}+2\pi+2} \approx 1.01868$ (and this value is attained when $\alpha = \pi/2$). In other words, our algorithm is provably off from the optimal by a multiplicative factor of at most 1.01868. This is also summarized in Figure 4.There is a lot of technical involvement for improving the performance of the Sweeping Algorithm for the problem. Figure 5 summarizes this improvement, while Figure 6 shows all the upper and lower bounds in the discussion above. Notably, the improvement we obtain is linear in the radius of the disk when α is bounded away from 0 or π.

Given that our optimization problem has limited information about the input, it may be seen as an online task. Notably, our algorithms may not be nearly optimal with respect to competitive analysis. The optimal offline algorithm (knowing the exact locations of treasure and exit) has cost $1 + 2\sin(a/2)$. Comparing this to the performance of our online algorithm for problem 1-TE$_{\geq}$ gives rise to competitive ratio that is decreasing with α, and ranging from $1+2\pi$, when $\alpha = 0$, to $\frac{5+\pi}{3} \approx 2.71386$, when $\alpha = \pi$. For problem 1-TE$_{=}$, the induced

competitive ratio exhibits very similar behavior. We leave it as an open problem as to whether the competitive ratio can be improved. That would be in addition to sharpening our bounds and to investigating our problem in other (continuous) geometric or discrete domains.

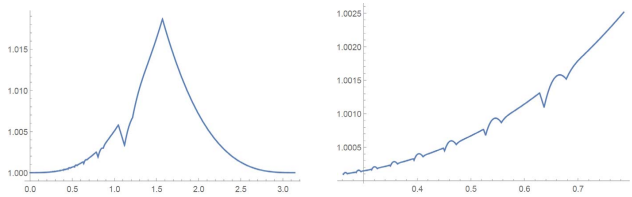

Fig. 4. The left-hand side plot is the ratio between the derived upper and lower bounds for problem 1-TE$_=$, for all values of α ranging from 0 to π. The right-hand side plot is a close-up for values of α between $\pi/12$ and $\pi/4$.

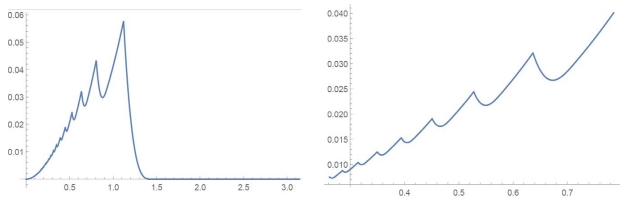

Fig. 5. The left-hand side plot is the difference between the naive Sweeping Algorithm and the derived upper bound for problem 1-TE$_=$, for all values of α ranging from 0 to π. The right-hand side plot is a close-up for values of α between $\pi/12$ and $\pi/4$.

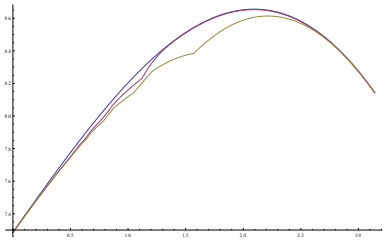

Fig. 6. Comparison between the cost of the naive sweeping algorithm (blue curve), the cost of Algorithm 2 (purple curve) and our lower bound (yellow curve).

References

1. R. Ahlswede and I. Wegener. *Search problems*. Wiley-Interscience, 1987.
2. S. Alpern. Find-and-fetch search on a tree. *Operations Research*, 59(5):1258–1268, 2011.
3. S. Alpern and S. Gal. *The theory of search games and rendezvous*. Springer, 2003.
4. R. Baeza Yates, J. Culberson, and G. Rawlins. Searching in the plane. *Information & Comp.*, 106(2):234–252, 1993.
5. R. Baeza-Yates and R. Schott. Parallel searching in the plane. *Computational Geometry*, 5(3):143–154, 1995.
6. A. Beck. On the linear search problem. *Israel Journal of Mathematics*, 2(4):221–228, 1964.
7. R. Bellman. An optimal search. *SIAM Review*, 5(3):274–274, 1963.
8. P. Berman. On-line searching and navigation. In A. Fiat and G. J. Woeginger, editors, *Online Algorithms: The State of the Art*, pages 232–241. Springer, 1998.
9. A. Bonato and R. Nowakowski. *The game of cops and robbers on graphs*. AMS, 2011.
10. M. Chrobak, L. Gasieniec, Gorry T., and R. Martin. Group search on the line. In *SOFSEM 2015*. Springer, 2015.
11. T. H. Chung, G. A. Hollinger, and V. Isler. Search and pursuit-evasion in mobile robotics. *Autonomous robots*, 31(4):299–316, 2011.
12. J. Czyzowicz, L. Gasieniec, T. Gorry, E. Kranakis, R. Martin, and D. Pajak. Evacuating robots from an unknown exit located on the perimeter of a disc. In *DISC 2014*. Springer, Austin, Texas, 2014.
13. J. Czyzowicz, K. Georgiou, E. Kranakis, L. Narayanan, J. Opatrny, and B. Vogtenhuber. Evacuating robots from a disc using face to face communication. In *CIAC 2015*. Springer, Paris, France, 2015.
14. J. Czyzowicz, E. Kranakis, D. Krizanc, L. Narayanan, J. Opatrny, and S. Shende. Wireless autonomous robot evacuation from equilateral triangles and squares. In *ADHOC-NOW 2015, Athens, Greece, June 29 - July 1, 2015, Proceedings*, pages 181–194, 2015.
15. E. D. Demaine, S. P. Fekete, and S. Gal. Online searching with turn cost. *Theoretical Computer Science*, 361(2):342–355, 2006.
16. S. Fekete, C. Gray, and A. Kröller. Evacuation of rectilinear polygons. In *Combinatorial Optimization and Applications*, pages 21–30. Springer, 2010.
17. B. Gluss. An alternative solution to the lost at sea problem. *Naval Res, Logistics Quarterly*, 8(1):117–122, 1961.
18. J. R. Isbell. Pursuit around a hole. *Naval Res. Logistics Quarterly*, 14(4):569–571, 1967.
19. J. S. Jennings, G. Whelan, and W. F. Evans. Cooperative search and rescue with a team of mobile robots. In *ICAR*, pages 193–200. IEEE, 1997.
20. M.-Y. Kao, Y. Ma, M. Sipser, and Y. Yin. Optimal constructions of hybrid algorithms. *J. Algorithms*, 29(1):142–164, 1998.
21. M.-Y. Kao, J. H. Reif, and S. R. Tate. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Information & Comp.*, 131(1):63–79, 1996.
22. E. Koutsoupias, C. Papadimitriou, and M. Yannakakis. Searching a fixed graph. In *ICALP 96*, pages 280–289. Springer, 1996.
23. P. Nahin. *Chases and Escapes: The Mathematics of Pursuit and Evasion*. Princeton University Press, 2012.
24. L. Stone. *Theory of optimal search*. Academic Press New York, 1975.