Deterministic Treasure Hunt in the Plane with Angular Hints

Sébastien Bouchard, Yoann Dieudonné, Andrzej Pelc, Franck Petit

Deterministic Treasure Hunt in the Plane with Angular Hints

Sébastien Bouchard
Sorbonne Université, CNRS, INRIA, LIP6, F-75005 Paris, France
sebastien.bouchard@lip6.fr

Yoann Dieudonné
Laboratoire MIS, Université de Picardie Jules Verne, Amiens, France
yoann.dieudonne@u-picardie.fr

Andrzej Pelc[1]
Département d'informatique, Université du Québec en Outaouais, Gatineau, Canada
pelc@uqo.ca

Franck Petit[2]
Sorbonne Université, CNRS, INRIA, LIP6, F-75005 Paris, France
franck.petit@lip6.fr

Abstract

A mobile agent equipped with a compass and a measure of length has to find an inert treasure in the Euclidean plane. Both the agent and the treasure are modeled as points. In the beginning, the agent is at a distance at most $D > 0$ from the treasure, but knows neither the distance nor any bound on it. Finding the treasure means getting at distance at most 1 from it. The agent makes a series of moves. Each of them consists in moving straight in a chosen direction at a chosen distance. In the beginning and after each move the agent gets a hint consisting of a positive angle smaller than 2π whose vertex is at the current position of the agent and within which the treasure is contained. We investigate the problem of how these hints permit the agent to lower the cost of finding the treasure, using a deterministic algorithm, where the cost is the worst-case total length of the agent's trajectory. It is well known that without any hint the optimal (worst case) cost is $\Theta(D^2)$. We show that if all angles given as hints are at most π, then the cost can be lowered to $O(D)$, which is optimal. If all angles are at most β, where $\beta < 2\pi$ is a constant unknown to the agent, then the cost is at most $O(D^{2-\epsilon})$, for some $\epsilon > 0$. For both these positive results we present deterministic algorithms achieving the above costs. Finally, if angles given as hints can be arbitrary, smaller than 2π, then we show that cost $\Theta(D^2)$ cannot be beaten.

2012 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems; G.2.1 Combinatorics

Keywords and phrases treasure hunt, deterministic algorithm, mobile agent, hint, plane

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

---

[1] This work was supported in part by NSERC discovery grant 8136 – 2013 and by the Research Chair in Distributed Computing of the Université du Québec en Outaouais.
[2] This work was performed within Project ESTATE (Ref. ANR-16-CE25-0009-03), supported by French state funds managed by the ANR (Agence Nationale de la Recherche).

42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:12
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Motivation. A tourist visiting an unknown town wants to find her way to the train station or a skier lost on a slope wants to get back to the hotel. Luckily, there are many people that can help. However, often they are not sure of the exact direction: when asked about it, they make a vague gesture with the arm swinging around the direction to the target, accompanying the hint with the words "somewhere there". In fact, they show an angle containing the target. Can such vague hints help the lost traveller to find the way to the target? The aim of the present paper is to answer this question.

The model and problem formulation. A mobile agent equipped with a compass and a measure of length has to find an inert treasure in the Euclidean plane. Both the agent and the treasure are modeled as points. In the beginning, the agent is at a distance at most $D > 0$ from the treasure, but knows neither the distance nor any bound on it. Finding the treasure means getting at distance at most 1 from it. In applications, from such a distance the treasure can be seen. The agent makes a series of moves. Each of them consists in moving straight in a chosen direction at a chosen distance. In the beginning and after each move the agent gets a hint consisting of a positive angle smaller than 2π whose vertex is at the current position of the agent and within which the treasure is contained. We investigate the problem of how these hints permit the agent to lower the cost of finding the treasure, using a deterministic algorithm, where the cost is the worst-case total length of the agent's trajectory. It is well known that the optimal cost of treasure hunt without hints is $\Theta(D^2)$. (The algorithm of cost $O(D^2)$ is to trace a spiral with jump 1 starting at the initial position of the agent, and the lower bound $\Omega(D^2)$ follows from Proposition 5.1 which establishes this lower bound even assuming arbitrarily large angles smaller than 2π given as hints.)

Our results. We show that if all angles given as hints are at most π, then the cost of treasure hunt can be lowered to $O(D)$, which is optimal. Our real challenge here is in the fact that hints can be angles of size *exactly* π, in which case the design of a trajectory always leading to the treasure, while being cost-efficient in terms of traveled distance, is far from obvious.

If all angles are at most β, where $\beta < 2\pi$ is a constant unknown to the agent, then we prove that the cost is at most $O(D^{2-\epsilon})$, for some $\epsilon > 0$. Finally, we show that arbitrary angles smaller than 2π given as hints cannot be of significant help: using such hints the cost $\Theta(D^2)$ cannot be beaten.

For both our positive results we present deterministic algorithms achieving the above costs. Both algorithms work in phases "assuming" that the treasure is contained in increasing squares centered at the initial position of the agent. The common principle behind both algorithms is to move the agent to strategically chosen points in the current square, depending on previously obtained hints, and sometimes perform exhaustive search of small rectangles from these points, in order to guarantee that the treasure is not there. This is done in such a way that, in a given phase, obtained hints together with small rectangles exhaustively searched, eliminate a sufficient area of the square assumed in the phase to eventually permit finding the treasure.

In both algorithms, the points to which the agent travels and where it gets hints are chosen in a natural way, although very differently in each of the algorithms. The main difficulty is to prove that the distance travelled by the agent is within the promised cost. In the case of the first algorithm, it is possible to cheaply exclude large areas not containing the treasure, and thus find the treasure asymptotically optimally. For the second algorithm, the agent eliminates smaller areas at each time, due to less precise hints, and thus finding the treasure costs more.

Due to lack of space, the details of one of the algorithms and proofs of several results are in the Appendix that is the full version of the paper.

Related work. The problem of treasure hunt, i.e., searching for an inert target by one or more mobile agents was investigated under many different scenarios. The environment where the treasure is hidden may be a graph or a plane, and the search may be deterministic or randomized. An early paper [4] showed that the best competitive ratio for deterministic treasure hunt on a line is 9. In [8] the authors generalized this problem, considering a model where, in addition to travel length, the cost includes a payment for every turn of the agent. The book [2] surveys both the search for a fixed target and the related rendezvous problem, where the target and the finder are both mobile and their role is symmetric: they both cooperate to meet. This book is concerned mostly with randomized search strategies. Randomized treasure hunt strategies for star search, where the target is on one of m rays, are considered in [13]. In [17, 19] the authors study relations between the problems of treasure hunt and rendezvous in graphs. The authors of [3] study the task of finding a fixed point on the line and in the grid, and initiate the study of the task of searching for an unknown line in the plane. This research is continued, e.g., in [12, 16]. In [18] the authors concentrate on game-theoretic aspects of the situation where multiple selfish pursuers compete to find a target, e.g., in a ring. The main result of [15] is an optimal algorithm to sweep a plane in order to locate an unknown fixed target, where locating means to get the agent originating at point O to a point P such that the target is in the segment OP. In [10] the authors consider the generalization of the search problem in the plane to the case of several searchers. Collective treasure hunt in the grid by several agents with bounded memory is investigated in [9, 14]. In [5], treasure hunt with randomly faulty hints is considered in tree networks. By contrast, the survey [7] and the book [6] consider pursuit-evasion games, mostly on graphs, where pursuers try to catch a fugitive target trying to escape.

2 Preliminaries

Since for $D \leq 1$ treasure hunt is solved immediately, in the sequel we assume $D > 1$. Since the agent has a compass, it can establish an orthogonal coordinate system with point O with coordinates $(0, 0)$ at its starting position, the x-axis going East-West and the y-axis going North-South. Lines parallel to the x-axis will be called horizontal, and lines parallel to the y-axis will be called vertical. When the agent at a current point a decides to go to a previously computed point b (using a straight line), we describe this move simply as "Go to b". A hint given to the agent currently located at point a is formally described as an ordered pair (P_1, P_2) of half-lines originating at a such that the angle clockwise from P_1 to P_2 (including P_1 and P_2) contains the treasure.

The line containing points A and B is denoted by (AB). A segment with extremities A and B is denoted by $[AB]$ and its length is denoted $|AB|$. Throughout the paper, a polygon is defined as a closed polygon (i.e., together with the boundary). For a polygon S, we will denote by $\mathcal{B}(S)$ (resp. $\mathcal{I}(S)$) the boundary of S (resp. the interior of S, i.e., the set $S \setminus \mathcal{B}(S)$). A rectangle is defined as a non-degenerate rectangle, i.e., with all sides of strictly positive length. A rectangle with vertices A, B, C, D (in clockwise order) is denoted simply by $ABCD$. A rectangle is *straight* if one of its sides is vertical.

In our algorithms we use the following procedure `RectangleScan`(R) whose aim is to traverse a closed rectangle R (composed of the boundary and interior) with known coordinates, so that the agent initially situated at some point of R gets at distance at most 1 from every point of it and returns to the starting point. We describe the procedure for a straight rectangle whose vertical side is not shorter than the horizontal side. The modification of the procedure for arbitrarily positioned rectangles is straightforward. Let the vertices of the rectangle R be A, B, C and D, where A is the North-West vertex and the others are listed clockwise. Let a be the point at which the agent starts the procedure.

The idea of the procedure is to go to vertex A, then make a snake-like movement in which consecutive vertical segments are separated by a distance 1, and then go back to point a. The agent ignores all hints gotten during the execution of the procedure. Suppose that the horizontal side of R has length m and the vertical side has length n, with $n \geq m$. Let $k = \lfloor m \rfloor$. Let $a_0, a_1, \ldots, a_k$ be points on the North horizontal side of the rectangle, such that $a_0 = A$ and the distance between consecutive points is 1. Let $b_0, b_1, \ldots, b_k$ be points on the South horizontal side of the rectangle, such that $b_0 = D$ and the distance between consecutive points is 1.

The pseudocode of procedure `RectangleScan`(R) is given in Algorithm 1.

Algorithm 1 Procedure `RectangleScan`(R)

if k is odd **then**
    for $i = 0$ **to** $k - 1$ **step** 2 **do**
        Go to a_i; Go to b_i;
        Go to b_{i+1}; Go to a_{i+1}
    end for
    Go to a
else
    for $i = 0$ **to** $k - 2$ **step** 2 **do**
        Go to a_i; Go to b_i;
        Go to b_{i+1}; Go to a_{i+1}
    end for
    Go to a_k; Go to b_k
    Go to a
end if

▶ Proposition 2.1. For every point p of the rectangle R, the agent is at distance at most 1 from p at some time of the execution of Procedure `RectangleScan`(R). The cost of the procedure is at most $5n \cdot \max(m, 2)$, where $n \geq m$ are the lengths of the sides of the rectangle.

3 Angles at most π

In this section we consider the case when all angles given as hints are at most π. Without loss of generality we can assume that they are all equal to π, completing any smaller angle to π in an arbitrary way: this makes the situation even harder for the agent, as hints become less precise. For such hints we show Algorithm `TreasureHunt1` that finds the treasure at cost $O(D)$. This is of course optimal, as the treasure can be at any point at distance at most D from the starting point of the agent.

For angles of size π, every hint is in fact a half-plane whose boundary line L contains the current location of the agent. For simplicity, we will code such a hint as $(L, right)$ or $(L, left)$, whenever the line L is not horizontal, depending on whether the indicated half-plane is to the right (i.e., East) or to the left (i.e., West) of L. For any non-horizontal line L this is non-ambiguous. Likewise, when L is horizontal, we will code a hint as (L, up) or $(L, down)$, depending on whether the indicated half-plane is up (i.e., North) from L or down (i.e., South) from L.

In view of the work on ϕ-self-approaching curves (cf. [1]) we first note that there is a big difference of difficulty between obtaining our result in the case when angles given as hints are *strictly smaller* than π and when they are *at most* π, as we assume. A ϕ-self-approaching curve is a planar oriented curve such that, for each point B on the curve, the rest of the curve lies inside a wedge of angle ϕ with apex in B. In [1], the authors prove the following property of these curves: for every $\phi < \pi$ there exists a constant $c(\phi)$ such that the length of any ϕ-self-approaching curve is at most $c(\phi)$ times the distance D between its endpoints. Hence, for angles ϕ strictly smaller than π, our result could possibly be derived from the existing literature: roughly speaking, the agent should follow a trajectory corresponding to any ϕ-self-approaching curve to find the treasure at a cost linear in D. Even then, transforming the continuous scenario of self-approaching curves to our discrete scenario presents some difficulties. However, the crucial problem is this: the result of [1] holds only when $\phi < \pi$ (the authors also emphasize that for each $\phi \geq \pi$, the property is false), and thus the above derivation is no longer possible for our purpose when $\phi = \pi$. Actually, this is the real difficulty of our problem: handling angles equal to π, i.e., half-planes.

We further observe that a rather straightforward treasure hunt algorithm of cost $O(D \log D)$, for hints being angles of size π, can be obtained using an immediate corollary of a theorem proven in [11] by Grünbaum: each line passing through the centroid of a convex polygon cuts the polygon into two convex polygons with areas differing by a factor of at most $\frac{5}{4}$. Suppose for simplicity that D is known. Starting from the square of side length $2D$, centered at the initial position of the agent, this permits to reduce the search area from P to at most $\frac{5P}{9}$ in a single move. Hence, after $O(\log D)$ moves, the search area is small enough to be exhaustively searched by procedure `RectangleScan` at cost $O(D)$. However, the cost of each move during the reduction is not under control and can be only bounded by a constant multiple of D, thus giving the total cost bound $O(D \log D)$. By contrast, our algorithm controls both the remaining search area and the cost incurred in each move, yielding the optimal cost $O(D)$.

The high-level idea of our Algorithm `TreasureHunt1` is the following. The agent acts in phases $j = 1, 2, 3, \dots$ where in each phase j the agent "supposes" that the treasure is in a straight square R_j centered at the initial position of the agent, and of side length 2^j. When executing a phase j, the agent successively moves to distinct points with the aim of using the hints at these points to narrow the search area that initially corresponds to R_j. In our algorithm, this narrowing is made in such a way that the remaining search area is always a straight rectangle. Often this straight rectangle is a strict superset of the intersection of all hints that the agent was given previously. This would seem to be a waste, as we are searching some areas that have been previously excluded. However, this loss is compensated by the ease of searching description and subsequent analysis of the algorithm, due to the fact that, at each stage, the search area is very regular.

During a phase, the agent proceeds to successive reductions of the search area by moving to distinct locations, until it obtains a rectangular search area that is small enough to be searched directly at low cost using procedure `RectangleScan`. In our algorithm, such a final execution of `RectangleScan` in a phase is triggered as soon as the rectangle has a side smaller than 4. If the treasure is not found by the end of this execution of procedure `RectangleScan`, the agent learns that the treasure cannot be in the supposed straight square R_j and starts the next phase from scratch by forgetting all previously received hints. This forgetting again simplifies subsequent analysis. The algorithm terminates at the latest by the end of phase $j_0 = \lceil \log_2 D \rceil + 1$, in which the supposed straight square R_{j_0} is large enough to contain the treasure. Hence, if the cost of a phase j is linear in 2^j, then the cost of the overall solution is linear in the distance D.

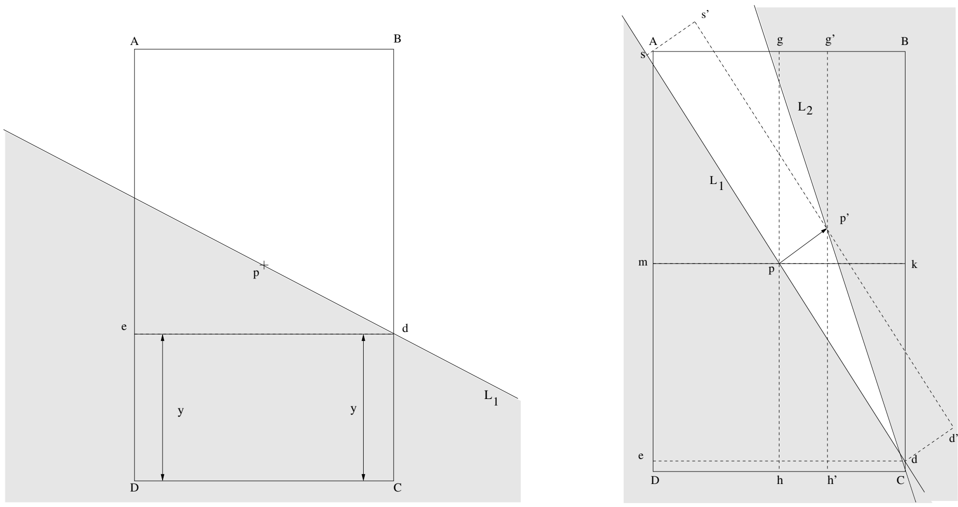
In order to give the reader deeper insights in the reasons why our solution is valid and has linear cost, we need to give more precise explanations on how the search area is reduced during a given phase $j \geq 2$ (when $j = 1$, the agent makes no reduction and directly scans the small search area using procedure `RectangleScan`). Suppose that in phase $j \geq 2$ the agent is at the center p of a search area corresponding to a straight rectangle R, every side of which has length between 4 and 2^j (note that this is the case at the beginning of the phase), and denote by A, B, C and D the vertices of R starting from the top left corner and going clockwise. In order to reduce rectangle R, the agent uses the hint at point p. The obtained hint denoted by (L_1, x_1) can be of two types: either a *good* hint or a *bad* hint. A good hint is a hint whose line L_1 divides one of the sides of R into two segments such that the length y of the smaller one is at least 1. A bad hint is a hint that is not good.

If the received hint (L_1, x_1) is good, then the agent narrows the search area to a rectangle $R' \subset R$ having the following three properties:

1. $R \setminus R'$ does not contain the treasure.
2. The difference between the perimeters of R and R' is $2y \geq 2$.
3. The distance from p to the center of R' is exactly $\frac{y}{2}$.

and then moves to the center of R'.

An illustration of such a reduction is depicted in Figure 1(a). The reduced search area R' is the rectangle $ABde$.



(a) A good hint $(L_1, right)$ (b) A bad hint $(L_1, right)$

Figure 1 In Figure (a) the agent received a good hint $(L_1, right)$ at the point p of a rectangular search area $ABCD$. In Figure (b) it received a bad hint $(L_1, right)$ at the point p and hence it moved to point p' and got a hint $(L_2, left)$. In both figures the excluded half-planes are shaded.

If the agent receives a bad hint, say $(L_1, right)$, at the center of a rectangular search area R, we cannot apply the same method as the one used for a good hint: this is the reason for the distinction between good and bad hints. If we applied the same method as before, we could obtain a rectangular search area R' such that the difference between the perimeters of R and R' is at least $2y$. However, in the context of a bad hint, the difference $2y$ may be very small (even null), and hence there is no significant reduction of the search area. In order to tackle this problem, when getting a bad hint at the center p of R, the agent moves to another point p' which is situated in the half-plane $(L_1, right)$ at distance 2 from p, perpendicularly to L_1. This point p' is chosen in such a way that, regardless of what is the second hint, we can ensure that two important properties described below are satisfied.

The first property is that by combining the two hints, the agent can decrease the search area to a rectangle $R' \subset R$ whose perimeter is smaller by 2 compared to the perimeter of R, as it is the case for a good hint, and such that $R \setminus R'$ does not contain the treasure. This decrease follows either directly from the pair of hints, or indirectly after having scanned some relatively small rectangles using procedure `RectangleScan`. In the example depicted in Fig. 1 (b), after getting the second hint $(L_2, left)$, the agent executes procedure `RectangleScan`$(ss'd'd)$ followed by `RectangleScan`$(gg'h'h)$ and moves to the center of the new search area R' that is the rectangle $Agpm$. Note that the part of R' not excluded by the two hints and by the procedure `RectangleScan` executed in rectangles $ss'd'd$ and $gg'h'h$ is only the small quadrilateral bounded by line L_2 and the segments $[AB]$, $[s'd']$ and $[gh]$. However, in order to preserve the homogeneity of the process, we consider the entire new search area R' which is a straight rectangle whose perimeter is smaller by at least 2, compared to that from R. This follows from the fact that no side of R has length smaller than 4. The agent finally moves to the center of R'.

The second property is that all of this (i.e., the move from p to p', the possible scans of small rectangles and finally the move to the center of R') is done at a cost linear in the difference of perimeters of R and R', as shown in the Appendix. The two properties together ensure that, even with bad hints, the agent manages to reduce the search area in a significant way and at a small cost. So, regardless of whether hints are good or not, we can show that the cost of phase j is in $\mathcal{O}(2^j)$ and the treasure is found during this phase if the initial square is large enough. The difficulty of the solution is in showing that the moves prescribed by our algorithm in the case of bad hints guarantee the two above properties, and thus ensure the correctness of the algorithm and the cost linear in D.

The details of the algorithm and its analysis are in the Appendix.

▶ **Theorem 1.** *Consider an agent A and a treasure located at distance at most D from the initial position of A. By executing Algorithm* `TreasureHunt1`*, agent A finds the treasure after having traveled a distance $\mathcal{O}(D)$.*

4 Angles bounded by $\beta < 2\pi$

In this section we consider the case when all hints are angles upper-bounded by some constant $\beta < 2\pi$, unknown to the agent. The main result of this section is Algorithm `TreasureHunt2` whose cost is at most $O(D^{2-\epsilon})$, for some $\epsilon > 0$. For a hint (P_1, P_2) we denote by $\overline{(P_1, P_2)}$ the complement of (P_1, P_2).

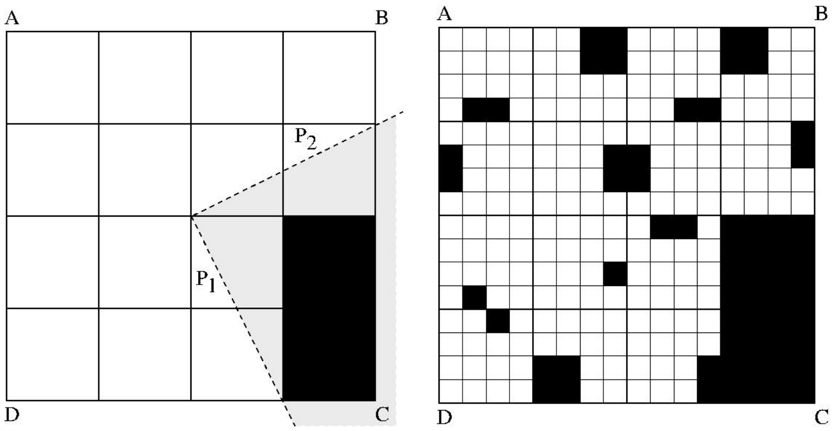
4.1 High level idea

In Algorithm `TreasureHunt2`, similarly as in the previous algorithm, the agent acts in phases $j = 1, 2, 3, \ldots$, where in each phase j the agent "supposes" that the treasure is in the straight square centered at its initial position and of side length 2^j. The intended goal is to search each supposed square at relatively low cost, and to ensure the discovery of the treasure by the time the agent finishes the first phase for which the initial supposed square contains the treasure. However, the similarity with the previous solution ends there: indeed, the hints that may now be less precise do not allow us to use the same strategy within a given phase. Hence we adopt a different approach that we outline below and that uses the following notion of tiling. Given a square S with side of length $x > 0$, $Tiling(i)$ of S, for any non-negative integer i, is the partition of square S into 4^i squares with side of length $\frac{x}{2^i}$. Each of these squares, called *tiles*, is closed, i.e., contains its border, and hence neighboring tiles overlap in the common border.

Let us consider a simpler situation in which the angle of every hint (P_1, P_2) is always equal to the bound β: the general case, when the angles may vary while being at most β, adds a level of technical complexity that is unnecessary to understand the intuition. In the considered situation, the angle of each excluded zone $\overline{(P_1, P_2)}$ is always the same as well. The following property holds in this case: there exists an integer i_β such that for every square S and every hint (P_1, P_2) given at the center of S, at least one tile of $Tiling(i_\beta)$ of S belongs to the excluded zone $\overline{(P_1, P_2)}$.

In phase j, the agent performs k steps: we will indicate later how the value of k should be chosen. At the beginning of the phase, the entire square S is white. In the first step, the agent gets a hint (P_1, P_2) at the center of S. By the above property, we know that $\overline{(P_1, P_2)}$ contains at least one tile of $Tiling(i_\beta)$ of S, and we have the guarantee that such a tile cannot contain the treasure. All points of all tiles included in $\overline{(P_1, P_2)}$ are painted black in the first step. This operation does not require any move, as painting is performed in the memory of the agent. As a result, at the end of the first step, each tile of $Tiling(i_\beta)$ of S is either black or white, in the following precise sense: a black tile is a tile all of whose points are black, and a white tile is a tile all of whose *interior* points are white.

In the second step, the agent repeats the painting procedure at a finer level. More precisely, the agent moves to the center of each white tile t of $Tiling(i_\beta)$ of S. When it gets a hint at the center of a white tile t, there is at least one tile of $Tiling(i_\beta)$ of t that can be excluded. As in the first step, all points of these excluded tiles are painted black. Note that a tile of $Tiling(i_\beta)$ of t is actually a tile of $Tiling(2i_\beta)$ of S. Moreover, each tile of $Tiling(i_\beta)$ of S is made of exactly 4^{i_β} tiles of $Tiling(2i_\beta)$ of S. Hence, as depicted in Figure 2, the property we obtain at the end of the second step is as follows: each tile of $Tiling(2i_\beta)$ of S is either black or white.



(*a*) At the end of a first step for a hint (P_1, P_2)

(*b*) At the end of a second step

Figure 2 White and black tiles at the end of the first and the second step of a phase, for square $S = ABCD$ and $i_\beta = 2$.

In the next steps, the agent applies a similar process at increasingly finer levels of tiling. More precisely, in step $2 < s \leq k$, the agent moves to the center of each white tile of $Tiling((s-1)i_\beta)$ of S and gets a hint that allows it to paint black at least one tile of $Tiling(s \cdot i_\beta)$ of S. At the end of step s, each tile of $Tiling(s.i_\beta)$ of S is either black or white. We can show that at each step s the agent paints black at least $\frac{1}{4^{i_\beta}}$th of the area of S that is white at the beginning of step s.

After step k, each tile of $Tiling(k \cdot i_\beta)$ of S is either black or white. These steps permit the agent to exclude some area without having to search it directly, while keeping some regularity of the shape of the black area. The agent paints black a smaller area than excluded by the hints but a more regular one. This regularity enables in turn the next process in the area remaining white. Indeed, the agent subsequently executes a brute-force searching that consists in moving to each white tile of $Tiling(k \cdot i_\beta)$ of S in order to scan it using the procedure `RectangleScan`. If, after having scanned all the remaining white tiles, it has not found the treasure, the agent repaints white all the square S and enters the next phase. Thus we have the guarantee that the agent finds the treasure by the end of phase $\lceil \log_2 D \rceil + 1$, i.e., a phase in which the initial supposed square is large enough to contain the treasure. The question is: how much do we have to pay for all of this? In fact, the cost depends on the value that is assigned to k in each phase j. The value of k must be large enough so that the distance travelled by the agent during the brute-force searching is relatively small. At the same time, this value must be small enough so that the the distance travelled during the k steps is not too large. A good trade-off can be reached when $k = \lceil \log_{4^{i_\beta}} \sqrt{2^j} \rceil$. Indeed, as highlighted in the proof of correctness, it is due to this carefully chosen value of k that we can beat the cost $\Theta(D^2)$ necessary without hints, and get a complexity of $\mathcal{O}(D^{2-\epsilon})$, where ϵ is a positive real depending on i_β, and hence depending on the angle β.

4.2 Algorithm and analysis

In this subsection we describe our algorithm in detail, prove its correctness and analyze its complexity. In the Appendix we define a function $index : (0, 2\pi) \longrightarrow \mathbb{N}^+$ that has the following properties, for any angle $0 < \alpha < 2\pi$.

1. For every square S and for every hint (P_1, P_2) of size $2\pi - \alpha$ obtained at the center of S, there exists a tile of $Tiling(index(\alpha))$ of S included in $\overline{(P_1, P_2)}$.
2. For every angle $\alpha' < \alpha$, we have $index(\alpha) \leq index(\alpha')$.

In the sequel, the integer $index(\alpha)$ is called the index of α. Algorithm 2 gives a pseudo-code of the main algorithm of this section. It uses the function `Mosaic` described in Algorithm 3 that is the key technical tool permitting the agent to reduce its search area. The agent interrupts the execution of Algorithm 2 as soon as it gets at distance 1 from the treasure, at which point it can "see" it and thus treasure hunt stops.

Algorithm 2 `TreasureHunt2`

```
IndexNew := 1
i := 1
loop
    repeat
        IndexOld := IndexNew
        IndexNew := Mosaic(i, IndexOld)
    until IndexNew = IndexOld
    i := i + 1
end loop
```

In the following, a square is called black if all its points are black. A square is called white if all points of its interior are white. (In a white square, some points of its border may be black).

▶ **Lemma 2.** *For any positive integers i and k, consider an agent executing function* `Mosaic`*(i,k) from its initial position O. Let S be the straight square centered at O with side of length 2^i. For every positive integer $j \leq \lceil \log_{4^k} \sqrt{2^i} \rceil$, at the end of the j-th execution of the first loop (lines 5 to 20) in* `Mosaic`*(i,k), each tile of $Tiling(jk)$ of S is either black or white.*

▶ **Lemma 3.** *For every positive integers i and k, a call to function* `Mosaic`*(i,k) has cost at most* $2^{i^{\frac{3+\log_{4^k}(4^k-1)}{2}}+2k+8}$.

Algorithm 3 Function `Mosaic`(i,k)

O:= the initial position of the agent
S:= the straight square centered at O with sides of length 2^i
Paint white all points of S
$IndexMax$:=k
for $j = 1$ **to** $\lceil \log_{4^k} \sqrt{2^i} \rceil$ **do**
  for all tiles t of $Tiling((j-1)k)$ of S **do**
    if t is white **then**
      Go to the center of t
      Let (P_1, P_2) be the obtained hint
      k':= index of $\overline{(P_1, P_2)}$
      if $k' > IndexMax$ **then**
        $IndexMax$:=k'
      end if
      if $IndexMax = k$ **then**
        for all tiles t' of $Tiling(k)$ of t such that $t' \subset \overline{(P_1, P_2)}$ **do**
          Paint black all points of t'
        end for
      end if
    end if
  end for
end for
if $IndexMax = k$ **then**
  for all tiles t of $Tiling(k(\lceil \log_{4^k} \sqrt{2^i} \rceil))$ of S **do**
    if t is white **then**
      Go to the center of t
      Execute `RectangleScan`(t)
    end if
  end for
end if
Go to O
return $IndexMax$

Let ψ be the index of $2\pi - \beta$. The next proposition follows from the aforementioned properties of the function *index*.

▶ Proposition 4.1. Let (P_1, P_2) be any hint. The index of $\overline{(P_1, P_2)}$ is at most ψ.

Using Lemmas 2, 3 and Proposition 4.1 we prove the final result of this section.

▶ **Theorem 4.** *Consider an agent A and a treasure located at distance at most D from the initial position of A. By executing Algorithm* `TreasureHunt2`*, agent A finds the treasure after having traveled a distance in $\mathcal{O}(D^{2-\epsilon})$, for some $\epsilon > 0$.*

Proof. We will use the following two claims.

▶ Claim 4.1. Let $i \geq 1$ be an integer. The number of executions of the repeat loop in the i-th execution of the external loop in Algorithm 2 is bounded by ψ.

Proof of the claim: Suppose by contradiction that the claim does not hold for some $i \geq 1$. So, the number of executions of the repeat loop in the i-th execution of the external loop in Algorithm 2 is at least $\psi + 1$. In each of these executions of the repeat loop, the agent calls function `Mosaic`$(i, *)$ exactly once. For all $1 \leq j \leq \psi + 1$ ($\psi \geq 1$, by definition of an index), denote by v_j the returned value of function `Mosaic`$(i, *)$ in the j-th execution of the repeat loop in the i-th execution of the external loop. Note that $v_1 \neq 1$: indeed, if $v_1 = 1$ the repeat loop would be executed exactly once, which would be a contradiction because it is executed at least $\psi + 1 \geq 2$ times.

In view of Algorithm 2 and Proposition 4.1, the returned value of `Mosaic`$(i, *)$ is a positive integer that is at most ψ. Since $v_1 \neq 1$, this implies that $\psi \geq 2$. Moreover, for all $2 \leq j \leq \psi$, we have $v_j \geq v_{j-1}$ (cf. lines 5-6 of Algorithm 2 and lines 4, 11-12 of Algorithm 3). Hence, there exists an integer $k \leq \psi$ such that $v_k = v_{k-1}$. However, according to Algorithm 2, this implies that the number of executions of the repeat loop in the i-th execution of the external loop is at most $k \leq \psi$. This is a contradiction which concludes the proof of the claim. ⋆

▶ Claim 4.2. The distance traveled by the agent before variable i becomes equal to $\lceil \log_2 D \rceil + 2$ in the execution of Algorithm 2 is $\mathcal{O}(D^{2-\epsilon})$, where $\epsilon = \frac{1}{2}(1 - \log_{4^\psi}(4^\psi - 1)) > 0$.

Proof of the claim: In view of the fact that the returned value of every call to function `Mosaic` in the execution of Algorithm 2 is at most ψ, it follows that in each call to function `Mosaic`$(*, k)$ the parameter k is always at most ψ. Hence, in view of Claim 4.1 and Lemma 3, as long as variable i does not reach the value $\lceil \log_2 D \rceil + 2$, the agent traveled a distance at most

$$\psi \cdot \sum_{i=1}^{\lceil \log_2 D \rceil + 1} 2^{i \frac{3 + \log_{4^\psi}(4^\psi - 1)}{2} + 2\psi + 8} \tag{1}$$

$$\leq \psi 2^{(\lceil \log_2 D \rceil + 1) \frac{3 + \log_{4^\psi}(4^\psi - 1)}{2} + 2\psi + 9} \tag{2}$$

$$\leq \psi 2^{2\psi + 12 + \log_{4^\psi}(4^\psi - 1)} 2^{(\log_2 D) \frac{3 + \log_{4^\psi}(4^\psi - 1)}{2}} \tag{3}$$

$$= \psi 2^{2\psi + 12 + \log_{4^\psi}(4^\psi - 1)} D^{2 - \frac{1}{2}(1 - \log_{4^\psi}(4^\psi - 1))} \tag{4}$$

By (4), the total distance traveled by the agent executing Algorithm 2 is $\mathcal{O}(D^{2-\epsilon})$ where $\epsilon = \frac{1}{2}(1 - \log_{4^\psi}(4^\psi - 1))$. Since ψ is a positive integer, we have $0 < \log_{4^\psi}(4^\psi - 1) < 1$ and hence $\epsilon > 0$. This ends the proof of the claim. ⋆

Assume that the theorem is false. As long as variable i does not reach $\lceil \log_2 D \rceil + 2$, the agent cannot find the treasure, as this would contradict Claim 4.2. Thus, in view of Claim 4.1, before the time τ when variable i reaches $\lceil \log_2 D \rceil + 2$ the treasure is not found. By Algorithm 2, this implies that during the last call to function `Mosaic` before time τ, the function returns a value that is equal to its second input parameter. This implies that during this call, the agent has executed lines 23 to 28 of Algorithm 3: more precisely, there is some integer x such that from each white tile t of $Tiling(x)$ of the straight square S that is centered at the initial position of the agent and that has sides of length $2^{\lceil \log_2 D \rceil + 1}$, the agent has executed function `RectangleScan`(t). Hence, at the end of the execution of lines 23 to 28, the agent has seen all points of each white tile of $Tiling(x)$ of S. Moreover, in view of Lemma 2, we know that the tiles that are not white, in $Tiling(x)$ of S, are necessarily black. Given a black tile σ of $Tiling(x)$, each point of σ is black, which, in view of lines 15 to 17 of Algorithm 3, implies that σ cannot contain the treasure. Since square S necessarily contains the treasure, it follows that the agent must find the treasure by the end of the last execution of function `Mosaic` before time τ. As a consequence, the agent stops the execution of Algorithm 2 before assigning $\lceil \log_2 D \rceil + 2$ to variable i and thus, we get a contradiction with the definition of time τ, which proves the theorem. ◀

5 Arbitrary angles

We finally observe that if hints can be arbitrary angles smaller than 2π then the treasure hunt cost $\Theta(D^2)$ cannot be improved in the worst case.

▶ Proposition 5.1. If hints can be arbitrary angles smaller than 2π then the optimal cost of treasure hunt for a treasure at distance at most D from the starting point of the agent is $\Omega(D^2)$.

6 Conclusion

For hints that are angles at most π we gave a treasure hunt algorithm with optimal cost linear in D. For larger angles we showed a separation between the case where angles are bounded away from 2π, when we designed an algorithm with cost strictly subquadratic in D, and the case where angles have arbitrary values smaller than 2π, when we showed a quadratic lower bound on the cost. The optimal cost of treasure hunt with large angles bounded away from 2π remains open. In particular, the following questions seem intriguing. Is the optimal cost linear in D in this case, or is it possible to prove a super-linear lower bound on it? Does the order of magnitude of this optimal cost depend on the bound $\pi < \beta < 2\pi$ on the angles given as hints?

References

1. O. Aichholzer, F. Aurenhammer, C. Icking, R. Klein, E. Langetepe and G. Rote, Generalized self-approaching curves, Discrete Applied Mathematics 109 (2001), 3-24
2. S.Alpern and S.Gal, The Theory of Search Games and Rendezvous, Kluwer Academic Publications, 2003.
3. R. Baeza-Yates, J. Culberson, and J. Rawlins, Searching the plane, Information and Computation 106 (1993), 234-252.
4. A. Beck, D.J. Newman, Yet more on the linear search problem, Israel J. Math. 8 (1970), 419-429.
5. L. Boczkowski, A. Korman, Y. Rodeh, Searching on trees with noisy memory, arXiv:1611.01403v2 [cs.DS] (2017).
6. A. Bonato and R.J. Nowakowski, The Game of Cops and Robbers on Graphs, American Mathematical Society, 2011.
7. T.H. Chung, G.A. Hollinger and V. Isler, Search and pursuit-evasion in mobile robotics: A survey, Auton. Robot 31 (2011), 299-316.
8. E. Demaine, S. Fekete, S. Gal, Online searching with turn cost, Theoretical Computer Science 361 (2006), 342 Ð 355.
9. Y. Emek, T. Langner, D. Stolz, J. Uitto, R. Wattenhofer, How many ants does it take to find the food? Theoretical Computer Science 608 (2015), 255-267.
10. G. M. Fricke, J. P. Hecker, A. D. Griego, L. T. Tran and Melanie E. Moses, A Distributed Deterministic Spiral Search Algorithm for Swarms, Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2016), 4430-4436.
11. B. Grünbaum, Partitions of mass-distributions and convex bodies by hyperplanes, Pacific J. Math. 10 (1960), 1257-1261
12. A. Jez and J. Lopuszanski, On the two-dimensional cow search problem, Information Processing Letters 109 (2009), 543 - 547.
13. M.Y. Kao, J.H. Reif, S.R. Tate, Searching in an unknown environment: an optimal randomized algorithm for the cow-path problem, Information and Computation 131 (1996), 63-79.
14. B. Keller, T. Langner, J. Uitto, R. Wattenhofer, Overcoming obstacles with ants, Proc. 19th International Conference on Principles of Distributed Systems (OPODIS 2015), 1-17.
15. E. Langetepe, On the Optimality of Spiral Search, Proc. 21st Ann. ACM-SIAM Symp. Disc. Algor. (SODA 2010), 1-12.
16. E. Langetepe, Searching for an axis-parallel shoreline, Theoretical Computer Science 447 (2012), 85-99.
17. A. Miller, A. Pelc, Tradeoffs between cost and information for rendezvous and treasure hunt, Journal of Parallel and Distributed Computing 83 (2015), 159-167.
18. K. Spieser and E. Frazzoli, The Cow-Path Game: A Competitive Vehicle Routing Problem, Proc. 51st IEEE Conference on Decision and Control (2012), 6513 - 6520.
19. A. Ta-Shma and U. Zwick, Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. ACM Transactions on Algorithms 10 (2014), 12:1-12:15.