The Marco Polo Problem: A Combinatorial Approach to Geometric Localization*

Ofek Gila† Michael T. Goodrich† Zahra Hadizadeh‡ Daniel S. Hirschberg† Shayan Taherijam†

Abstract

We introduce and study the ***Marco Polo problem***, which is a combinatorial approach to geometric localization. In this problem, we are told there are one or more points of interest (POIs) within distance n of the origin that we wish to localize. Given a mobile search point, Δ, that is initially at the origin, a localization algorithm is a strategy to move Δ to be within a distance of 1 of a POI. In the combinatorial localization problem we study, the only tool we can use is reminiscent of the children's game, "Marco Polo," in that Δ can issue a ***probe*** signal out a specified distance, d, and the search algorithm learns whether or not there is a POI within distance d of Δ. For example, we could imagine that POIs are one or more hikers lost in a forest and we need to design a search-and-rescue (SAR) strategy to find them using radio signal probes to a response device that hikers carry. Unlike other known localization algorithms, probe responses do not inform our search algorithm of the direction or distance to a POI. The optimization problem is to minimize the number of probes and/or POI responses, as well as possibly minimizing the distance traveled by Δ. We describe a number of efficient combinatorial Marco Polo localization strategies and we analyze each one in terms of the size, n, of the search domain. Moreover, we derive strong bounds for the constant factors for the search costs for our algorithms, which in some cases involve computer-assisted proofs. We also show how to extend these strategies to find all POIs using a simple, memoryless search algorithm, traveling a distance that is $\mathcal{O}(\log k)$-competitive with the optimal traveling salesperson (TSP) tour for k POIs.

1 Introduction

In the children's game, "Marco Polo," a group of children are playing in a swimming pool. One player is chosen as "it," who closes their eyes and tries to find and tag one of the other players. The "it" player periodically calls out "Macro" and the other players who can hear this call must respond with "Polo." The "it" player moves based on this "Marco-Polo" call-and-response protocol until getting close enough to another player to tag them, which ends this player's turn being "it." See, e.g., [20].

In this paper, we introduce and study the ***Marco Polo problem***, which is a combinatorial approach to geometric localization motivated from the "Marco Polo" children's game. In this problem, we start with a search point, Δ, at the origin, with one or more points of interest (POIs) at a distance n from the origin and our goal is to move Δ to be within distance 1 of a POI.[1] We may periodically send probes out a specified distance, d, and we learn whether or not there is a POI within distance d from Δ. Optimization goals include minimizing the number of probes and minimizing the total distance traveled by Δ.

We can motivate the Marco Polo problem, for example, from a search-and-rescue (SAR) scenario. Suppose a hiker, who we'll call "Alice," is lost and stationary at a point of interest (POI) in a large forest and we would like to find her using an efficient SAR strategy. Assume Alice has a wireless device, similar to an Apple AirTag, which can respond to probes sent from a searcher, Δ. In particular, suppose Δ can send a probe at a given power level, which sends an omni-directional signal out to a known radius (depending on the power level), and if Alice is present inside the circle determined by this probe, then Δ will receive a positive response. Such probes use up power, however, both for Δ and for Alice's tracking device; hence, the goal is to devise a sequence of probes that minimizes the number of probes needed to locate Alice to a specified accuracy. (See Figure 1.)

We also consider generalizations of this problem, such as if there were multiple POIs. One can imagine other applications besides search and rescue for the combinatorial searching problem, including locating animals wearing tracking collars, finding radioactive sources, or identifying anomalous readings in wireless sensor networks.

Related Prior Work. Although we are not familiar with any prior work on the Marco Polo problem itself,

*This research supported in part by NSF grant 2212129.

†University of California, Irvine, {ogila,goodrich,dhirschb,staherij}@uci.edu

‡University of Rochester, zhadizadeh99@gmail.com

[1] This formulation is made without loss of generality, as we could just as easily normalize the search problem so that there is a POI within distance 1 of Δ and we are interested in moving Δ to be within distance ε of a POI, for a given $\varepsilon > 0$.

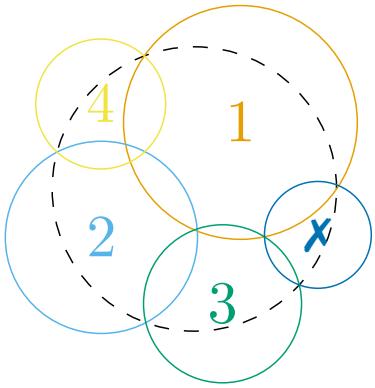


Figure 1: An example sequence of probes for an instance of the Marco Polo problem. In this example, the first four probes are negative and the POI (marked with an ✗) is found to within distance 1 on the fifth probe.

our work nevertheless falls into a rich area of study known as ***localization algorithms***; see, e.g., the survey by Han, Xu, Duong, Jiang, and Hara [12]. We discuss how our work compares to a wealth of existing prior work in Section A.

Problem Definition. In the ***Marco Polo problem***, there are $k \geq 1$ points of interest (POIs) with unknown positions, with at least one that is within a distance, n, of a point, O, called the ***origin***, which is the initial position of a mobile search point, Δ. A ***probe***, $p(x, y, d)$, is a query that asks if there are any POIs within distance d of the current location, (x, y), for Δ. The goal is to design a search strategy to move Δ to be within a distance of 1 of a POI. Given this setup, there are a number of possible constraints that define instances of the Marco Polo problem, including:

- A search algorithm can be ***incremental***, which finds all k POIs one at a time, or ***coordinated***, which finds multiple POIs in a coordinated fashion.
- An algorithm is ***memoryless*** with respect to the current search area if its state is restricted to the area determined by the previous successful probe.

There are a number of metrics we can use to measure the effectiveness of the strategy, including the following:

- $P(n)$: the number of probes issued.
- $R_{\max}$: the maximum number of times a POI responds to a probe.
- $D(n)$: the total distance traveled by Δ.

These efficiency measures can conflict, of course, in that a strategy that minimizes, say, $R_{\max}$, may have poor bounds for $P(n)$ and/or $D(n)$. Such a trade-off may nevertheless be worthwhile, however, such as in an SAR scenario where the batteries are running out in a hiker's device and responses are costly.

Our Results. In this paper, we provide a number of efficient algorithms for solving instances of the Marco Polo problem. In Section D we show a simple double-binary-search algorithm that uses $2\lceil \log n \rceil + \mathcal{O}(1)$ probes,[2] but which is based on unrealistic assumptions. Instead, under more realistic assumptions about conditions regarding the search space, we provide a sequence of algorithms, starting with simple algorithms based on hexagon geometries, which we call "hexagonal algorithms," and progressing to more sophisticated recursive strategies based on progressively shrinking probes at each level of recursion. This ultimately results in an algorithm that makes at most $3.34\lceil \log n \rceil$ probes using a monotonically spiraling search strategy at each recursive level. Using computer-assisted proof techniques, we then show that it is possible to find a POI using $2.53\lceil \log n \rceil$ probes. Although this strategy sacrifices the simplicity of a monotonically spiraling search, it performs competitively with our proven lower bound of $2.4\lceil \log n \rceil$ probes for progressive shrinking algorithms. We also provide various algorithms that reach different trade-offs between the number of probes made and the total distance traveled by Δ, include one algorithm that travels a total distance of at most $6.02n$, which is less than the circumference of the original search area. We then provide a family of algorithms that are able to restrict the total number of POI responses, $R_{\max}$, to any desired value from 1 to $\lceil \log n \rceil$ while minimizing the number of probes made. Finally, we present a strategy to extend our incremental algorithms to find all POIs while traveling a distance that is $\mathcal{O}(\log k)$-competitive with an optimal traveling salesperson (TSP) tour. We include experiments supporting all our results in Section F.

2 Finding One POI

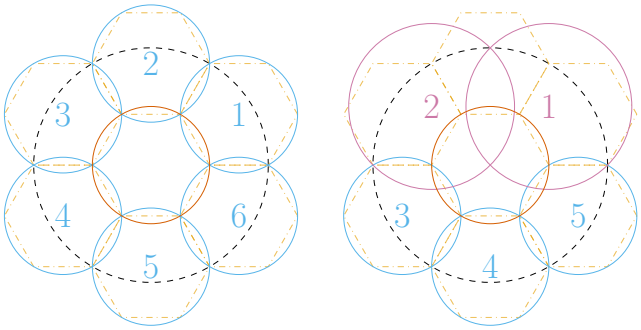
We describe a series of progressively more efficient algorithms that minimize the number of probes needed for finding a single POI. We assume that there may be multiple POIs, either within the search region of radius n or outside of it, but we are initially interested only in finding one of them. Later we will discuss how to extend these algorithms to find all POIs.

Hexagonal Algorithms. Our first algorithms, which we call ***hexagonal algorithms***, are defined in terms of a tiling of our search area with hexagons of radius $n/2$. There are seven such hexagons, which can each be probed with radius-$n/2$ probes until a probe succeeds, which then allows us to make a recursive call to an $n/2$-sized subproblem. Such an algorithm will take $7\lceil \log n \rceil$ probes in the worst case where POIs are always in the last hexagon probed in each recursive level. We can improve this to $6\lceil \log n \rceil$ probes by not probing the last hexagon, since a POI ***must*** be there

[2] All of the logarithms used in this paper are base 2.

if the other six probes fail. We refer to this hexagonal algorithm as Algorithm 1.

There is a better hexagonal algorithm, however, which involves first probing the upper two quadrants, which can be done with radius $n/\sqrt{2}$ probes, eliminating 3 hexagons, and then probing 3 of the 4 remaining hexagons as before. We refer to this modified hexagonal algorithm as Algorithm 2. See Figure 2. If POIs are in one of the two larger probes, we reduce the problem less than before, only by a factor of $\sqrt{2}$, but with fewer probes (at most 2). This turns out to be a better tradeoff, so that in the worst case where POIs are all in the last hexagon probed at each level and our algorithm makes at most $5\lceil \log n \rceil$ probes.



Algorithm 1

$P(n) \le 6\lceil \log n \rceil$
$D(n) \le 10.39n$
$R_{\max} \le \lceil \log n \rceil$

Algorithm 2

$P(n) \le 5\lceil \log n \rceil$
$D(n) \le 8.81n$
$R_{\max} \le 2\lceil \log n \rceil$

Figure 2: Two simple hexagonal algorithms. Algorithm 1 performs probes of radius $n/2$ along the center of each of the 6 outer hexagons, while Algorithm 2 first performs two radius $n/\sqrt{2}$ probes in the upper two quadrants before performing the remaining probes of Algorithm 1.

Distance Traveled. To determine the total distance traveled by Δ, $D(n)$, in any probe-sequence algorithms, including our hexagonal algorithms, let us consider the first layer of probes, i.e., those performed when the search radius is n. In particular, consider the k-th probe (starting from 1) as having radius $r_k = \rho_k n$, where ρ_k represents the proportionality factor for the size of each probe relative to the (current) search radius. Let d_k denote the distance Δ travels to perform the k-th probe. Assuming we always first succeed on our k-th probe, the total distance traveled is determined by $D(n) = d_k + D(\rho_k n)$, where we assume that $D(n) = b_k n$, for some constant b_k. Solving this, we find that if indeed the first probe that succeeds is always the k-th probe, then $b_k = d_k/(1-\rho_k)$. Using this, we determine that:

$$D(n) \le \max_k \frac{d_k}{1-\rho_k} n. \tag{1}$$

To determine a good bound for this maximum, we use a computer-assisted proof technique to compute this value for each algorithm.[3] The maximum number of responses is achieved when the POI is always inside the largest probe, $\rho_{\max}$, resulting in:

$$R_{\max} \le \lceil \log_{1/\rho_{\max}} n \rceil \le -\frac{1}{\log \rho_{\max}} \lceil \log n \rceil. \tag{2}$$

Progressively Shrinking Probes. As shown in Algorithm 2, we may be able to achieve better results by probing from differently sized circles before ever receiving a positive response. Indeed, we can do better by using probes that get progressively smaller, so that if we spend many probes to be able to recurse to a smaller search area, we should at least reduce the remaining area by a larger factor. The question remains how to choose the sizes of the probes.

Let the total number of probes required to find a POI starting with a search area of radius n be $P(n)$, and let the k-th probe (starting from 1) have radius $r_k = \rho_k n$, where ρ_k represents the proportionality factor for the size of each probe relative to current search radius. Assuming we always first succeed on our k-th probe, $P(n)$ is determined by the recurrence relation, $P(n) = k + P(\rho_k n)$, where $P(n) = c_k \log n$, for some constant, $c_k > 0$, and our total number of probes will be $k + P(\rho_k n)$, resulting in

$$\begin{aligned} P(n) &= k + P(\rho_k n) = k + c_k \log(\rho_k n) \\ &= k + c_k \log \rho_k + c_k \log n \\ &= k + c_k \log \rho_k + P(n). \end{aligned}$$

Hence, $0 = k + c_k \log \rho_k \implies c_k = -\frac{k}{\log \rho_k}$.

In the worst case, all POIs will be in the k-th probe such that the total number of probes is maximized, i.e., such that c_k is maximized, so we pick c_k such that it is the same for all k. This is done by setting $\rho_k = \rho_1^k$, where ρ_1 is the proportionality constant of the first probe, resulting in the overall number of probes being[4]

$$P(n) \le -\frac{1}{\log \rho_1} \log n. \tag{3}$$

Note that our equations for the maximum number of probes and responses, Equations (2) and (3), respectively, are nearly identical—the latter occurring when the POI is always in the first probe.

It is worth noting that all the above calculations make no assumptions about the correctness of the algorithm, i.e., the ability of the algorithm to always find a POI when one exists. Any *correct* algorithm must be able to cover the entire search area, including its perimeter, a fact that we will use to derive a lower bound on the number of probes required to find a POI.

Further, unlike the calculations for Equation (3) which assumed that the final probe locating the POI

[3] We provide pseudocode for this algorithm in Section B.

[4] The inequality arises from the omission of the last probe.

has radius which is exactly 1, it is possible for the final probe to have a smaller radius, i.e., that the algorithm obtained some excess precision. As we will discuss in more detail in Section C, this excess precision results in at most an additional constant number of probes, where the constant is dependent on the maximum number of probes the algorithm performs per layer, which we refer to as $k_{\max}$, obtaining:

Lemma 1 *A correct progressively shrinking algorithm may require at most* $-\frac{1}{\log \rho_1} \log n + \mathbf{k}_{max} - \mathbf{1}$ *probes to find a POI in a search area of radius* n.

The proof, as well as examples and more details are provided in Section C. For simplicity, we will ignore this additional constant in the main text.

A Lower Bound for Progressive Shrinking. With this result in mind, we can determine a lower bound on the number of probes required to find a POI. Since it is always possible for all POIs to be along the area's perimeter, any correct algorithm must at least probe the perimeter of the search area. To maximize the perimeter coverage of each probe, we place it such that its diameter is a chord of the circle, as shown in Figure 3, and determine the minimum value of ρ_1 required to probe the entire perimeter of the circle to be approximately 0.74915. This results in a lower bound of $P(n) > 2.40001 \log n$ probes.[5]

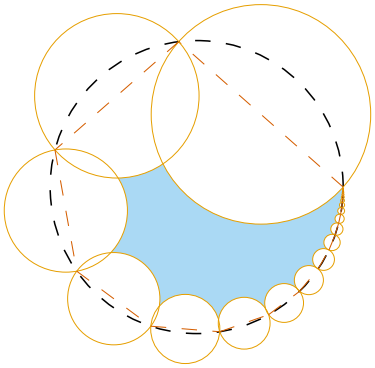
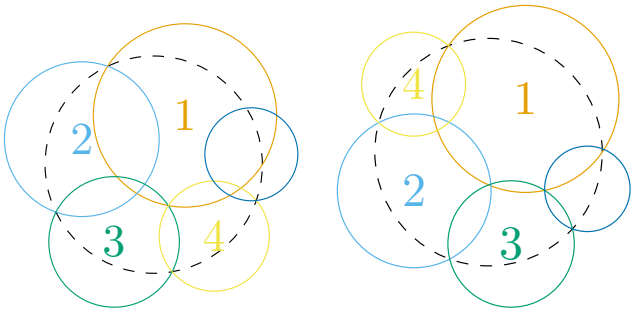


Figure 3: The optimal placement of progressively decreasing probes of radius proportional to ρ_1^k in order to cover the circumference of the search area. ρ_1 must be at least 0.74915 to fully cover the perimeter, as shown. There exists uncovered search area, depicted in blue.

Chord-Based Shrinking Algorithms. The placement strategy for Δ used to show the lower bound of roughly $2.4 \log n$ probes for progressive shrinking algorithms is not a valid approach for an upper-bound for an algorithm, due to all the internal uncovered area, see Figure 3. Nevertheless, the approach of placing diameters of each probe as chords of a circle can work with large enough probes, which we can determine by tuning ρ_1. We refer to our next algorithm, which is a progressive shrinking algorithm, as Algorithm 3. In this algorithm, we numerically determine, using a computer-assisted proof,[6] that the minimum value of ρ_1 that leaves no uncovered area is approximately 0.844, reducing the number of probes to $P(n) < 4.08 \log n$. See Figure 4 (left). Δ's route maintains the property that it is monotonic in a counterclockwise orientation.

However, if we allow for nonmonotonic routes, we can place the two largest probes side by side and alternate the next two probes on either side, which leads to significant improvements, which we refer to as Algorithm 4. This algorithm is able to further reduce the number of probes to $P(n) < 3.54 \log n$, and is depicted in Figure 4 (right).



Algorithm 3 | Algorithm 4

$P(n) < 4.08 \log n$, $D(n) \leq 6.95n$ | $P(n) < 3.54 \log n$, $D(n) \leq 9.31n$

Figure 4: Algorithms 3 and 4 both place probes such that their diameters are chords of the search area circle. Algorithm 3 places the probes in order of decreasing size going counter-clockwise, while Algorithm 4 places the probes such that they overlap as little as possible.

An optimization can be made to the total distance traveled by Δ for Algorithms 3 and 4. If a POI is determined to be in the last area, because previous probes in a recursive level are all negative, Δ can travel directly to the center of the first probe within the next recursive layer instead of the center of the last probe, since the last probe does not need to be performed. Accordingly, Algorithms 3 and 4 tradeoff the number of probes for flight distance. See Figure 4.

Higher-count Monotonic-path Algorithms. Both Algorithms 3 and 4 use very few probes (i.e., 5) at each recursive layer, while we know from Equation (3) that we can introduce more probes with geometrically decreasing radii without increasing $P(n)$. However, decreasing the value of ρ_1 in Algorithm 4 will not only introduce a gap in the perimeter but would also introduce an internal gap, which would require another placement scheme entirely to fill. Our next idea is to

[5] The exact coefficient c can be determined numerically by solving the following equation: $\sum_{k=1}^{\infty} \sin^{-1} 2^{-\frac{k}{c}} = \pi$, which we approximated using Wolfram Mathematica.

[6] The code determining ρ_1 and the maximum distance traveled is available at https://github.com/ofekih/DroneSearching.

begin with one large central probe before placing the remaining probes along the perimeter monotonically as in Algorithm 3, which should also have good flight-path performance for Δ. Intuitively, the large central probe greatly reduces the probe radius required to avoid internal gaps, allowing for more probes to be placed. Indeed, by placing the remaining probes such that their diameters are chords of the search area, as was done in Algorithm 3, leads to Algorithm 5, which uses up to 8 probes at each recursive level. While this algorithm improves upon Algorithm 3, requiring only $P(n) < 3.83 \log n$ probes, and improved flight distance, it performs worse than Algorithm 4 in its total number of probes. See Figure 6. In order to take advantage of even more probes, we observe that we cover the circumference of the search area at a much faster rate than the circumference of the central probe. In other words, if we reduce the probe radii, we would still be able to cover the search radius circumference, but we would introduce internal gaps between the outer probes and the central probe. Ideally, we would like the rate at which they cover the inner and the outer circumferences to be the same, such that the chords made with the outer and inner circles cover the same angle. The geometric reasoning is shown in Figure 5.

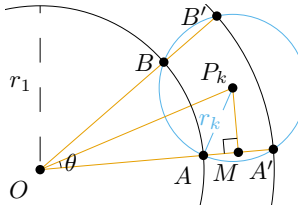
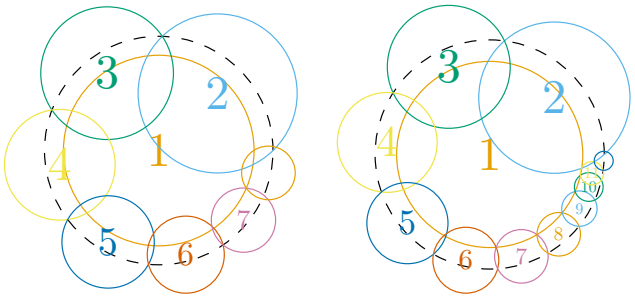


Figure 5: A diagram showing the relationship between the k-th circle, centered at P_k, the first circle, centered at O, and the search radius, also centered at O.

In particular, we must determine at what position, P_k, to place the center C_k of the k-th probe with radius r_k. For simplicity, we assume that the search area is a unit circle centered at the origin, O, and that the first probe, C_1, has radius $r_1 < 1$. Let A, B, A', and B' refer to the points where C_k intersects C_1 and the search area, respectively, and let M be the midpoint of $\overline{AA'}$. Covering both the inner and outer circumferences at the same rate implies that $\angle AOP_k = \angle A'OP_k$, i.e., that $\overline{OA}$ and $\overline{OA'}$ are colinear. Since $\triangle AA'P_k$ is isosceles, we know that $\overline{P_kM}$ is perpendicular to $\overline{AA'}$ so $|\overline{P_kM}| = \sqrt{r_k^2 - (\frac{1-r_1}{2})^2}$ and consequently $\theta = \arctan(\frac{\sqrt{r_k^2-(\frac{1-r_1}{2})^2}}{1-\frac{1-r_1}{2}})$. Thus, by moving the outer probes inward, towards the origin, to their new centers as described above, we are able to "turn it up to 11" and achieve Algorithm 6. See Figure 6.

Darting Non-Monotonic Algorithms. Up to this point, our best two algorithms for minimizing $P(n)$ are

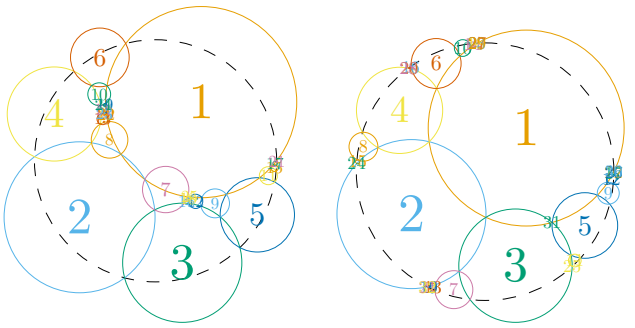


Algorithm 5
$P(n) < 3.83 \log n$
$D(n) \leq 6.72n$

Algorithm 6
$P(n) < 3.34 \log n$
$D(n) \leq 6.02n$

Figure 6: Algorithms 5 and 6 both perform probes counter-clockwise along the circumference of a central probe. Algorithm 5 places the probes so that diameters are chords of the search area, while Algorithm 6 balances the coverage rate of the inner and outer circumferences. Out of all our algorithms, Δ travels the least distance (in the worst case) using Algorithm 6.

Algorithm 4, which is non-monotonic and uses its few probes very efficiently, and Algorithm 6, which is monotonic with a counterclockwise spiral of probes such that, despite having significant overlap, it is able to squeeze in nearly three times as many probes and achieve better performance. The question remains whether it is possible to achieve even better query performance at the expense of monotonicity, giving up on optimizing Δ's flight distance so as to achieve even better probe complexity. We refer to such algorithms as being ***darting algorithms*** and discuss them in Section E. The best two darting algorithms are highlighted in Figure 7.



Algorithm 7
$P(n) < 2.93 \log n$
$D(n) \leq 25.8n$

Algorithm 8
$P(n) < 2.53 \log n$
$D(n) \leq 45.4n$

Figure 7: Algorithms 7 and 8 both use computer-assisted probe placement to efficiently cover the search area. Algorithm 7 begins with Algorithm 4, removing the final probe, while Algorithm 8 uses a differential evolution algorithm to place the initial six probes. The remaining probes are placed greedily as discussed in an appendix. Algorithm 8 achieves our best probe results.

Theorem 2 *Progressive shrinking algorithms for Δ searching a circular region of radius n have a lower bound of $2.4 \log n$ for $P(n)$, and we can achieve upper bounds as shown in Figures 2, 4, 6, and 7.*

Reducing The POIs' Responses. Our exploration has so far focused on reducing the total number of probes, $P(n)$, required to find a POI. In this section, we consider the case where POIs are limited in their number of responses $R_{\max}$ to probes, e.g., due to battery constraints. Namely, consider a scenario where $R_{\max}$ is a fixed value where $R_{\max} \geq 1$. Our goal is to design a search strategy that minimizes the number of probes, $P(n)$, while ensuring that the number of responses from the POIs does not exceed $R_{\max}$. Recalling from Equation (2), the amount of responses is determined by the size of the largest probe at each recursive layer, so there is no benefit to using differently sized probes. We present a family of hexagonal algorithms which probe the search space using hexagonal lattices with L layers of hexagonal rings. See Figure 8.

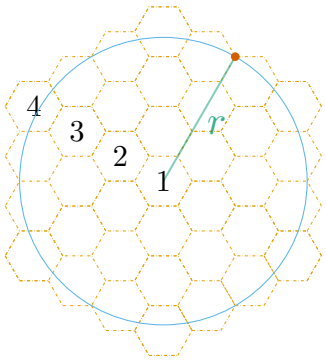


Figure 8: A hexagonal lattice with $L = 4$ layers of rings.

Increasing the number of rings increases the number of hexagons per lattice, which in turn reduces each hexagon's size. These smaller hexagons require smaller probes, reducing the number of responses from the POI but increasing the (worst case) total number of probes. Algorithm 1 can be thought of as one such algorithm where $L = 2$. We describe our family of algorithms by the following routine:

1. Cover the search area with an L-layer lattice.
2. Sequentially probe each hexagon in the lattice until receiving a positive response.
3. Repeat until reaching an area with radius 1.

Theorem 3 *If a POI is only allowed to respond at most $1 \leq R_{max} \leq \lceil \log n \rceil$ times, then a hexagonal algorithm is able to find them using at most*

$$P(n) \leq 6R_{max}\binom{\lceil \frac{2n^{\frac{1}{R_{max}}}+2}{3} \rceil}{2} \text{ probes,} \tag{4}$$

by using a hexagonal lattice with

$$L = \lceil \frac{2n^{\frac{1}{R_{max}}}+2}{3} \rceil \text{ layers.} \tag{5}$$

Fact 4 *A hexagon with side length s is covered by a circumscribed circle with radius s.*

Lemma 5 *A circle with radius r can be covered by an L-layer lattice of hexagons with side length $s = \frac{2r}{3L-2}$.*

Proof. See Section C. □

Fact 6 *An L-layer lattice has $1+6\binom{L}{2}$ hexagons, determined by the L-th centered hexagonal number.*

Proof. (of Theorem 3) During step 1, we cover the radius r search area with a lattice of hexagons of side length s, where $s = \frac{2r}{3L-2}$ from Lemma 5. From Fact 4, we probe each of these hexagons with a circle of radius s, reducing the search area of the next recursive layer by a factor of $\frac{3L}{2} - 1$. As the lattice grows, we reduce the search area by a greater factor. In order to finally reach a circle of radius 1, we require $\lceil \log_{\frac{3L}{2}-1}(n) \rceil$ recursive rounds, each requiring one response from the POI. Solving for L, we obtain Equation (5). Finally, since we can probe all the hexagons but one per round, we probe at most $6\binom{L}{2}$ hexagons per round (see Fact 6), and since there are $R_{\max}$ rounds, the total number of probes is bounded by $P(n) \leq 6R_{\max}\binom{L}{2}$. □

Corollary 7 *The total number of probes required to find a POI is bounded as follows:*

1. *If $R_{max} = 1$, then $P(n) \leq \frac{4n^2}{3} + 6n + 6 = \mathcal{O}(n^2)$.*
2. *If $R_{max} = 2$, then $P(n) \leq \frac{8n}{3} + 12\sqrt{n} + 12 = \mathcal{O}(n)$.*
3. *If $R_{max} = \lceil \log n \rceil$, then $P(n) \leq 6\lceil \log n \rceil$.*[7]

3 Finding All POIs

Once one POI is found, we shut off the tracking device so that it stops responding to Δ's probes, yet the question remains—how should Δ search for the rest of the POIs? Since Δ is stateless with respect to its current search area, no knowledge is gained about other POIs from the search for the first POI. Even if Δ was able to retain its search path so far, the result of probes are binary; any previous positive probe result cannot be relied on. And even if the probe were able to determine the exact *quantity* of POIs within the area, it is possible for them to be in different search areas early on, resulting in the probe needing to perform its search almost entirely from scratch. This can trivially happen if the POIs are far from each other, but may also happen if the POIs are

[7] This follows from the fact that $n^{\frac{1}{\lceil \log n \rceil}} \leq 2$.

close to each other but on opposite sides of a search area boundary. Does there exist a ***coordinated*** search strategy that performs better than the *incremental* strategy of independently searching for each POI?

Let us assume we have a search algorithm $\mathcal{A}(n)$ that is able to find a single POI in $P(n) \leq c\lceil \log n \rceil$ probes, traveling a distance of $D(n) \leq dn$. We use $\mathcal{A}$ as a subroutine of the following strategy:

1. Find an arbitrary POI using $\mathcal{A}(n)$.
2. Shut off the tracking device of the found POI.
3. Without moving Δ, re-probe the area at radius 2, 4, 8, etc., until a probe returns a positive result (i.e., another POI is found).
4. Invoke $\mathcal{A}$ using this new radius to find another POI.
5. Repeat steps 2 to 4 until all POIs are found.

Assuming there are k POIs, we have:

Theorem 8 *The total number of probes (P_{tot}) and the total distance traveled by Δ (D_{tot}) during the memoryless search algorithm for all k POIs is at most:*

$$P_{tot} \leq c\lceil \log n \rceil + (c+1)(k-1)\lceil \log \overline{e} \rceil,$$
$$D_{tot} \leq dn + 2dE,$$

where $E < OPT(\lceil \log k \rceil + 1)$, $\overline{e} = \frac{E}{k-1}$, and OPT is the optimal tour length for the traveling salesperson problem (TSP) on the k POIs.

Proof. See Section C. □

References

[1] Esther M Arkin, Michael A Bender, Sándor P Fekete, Joseph SB Mitchell, and Martin Skutella. The freeze-tag problem: how to wake up a swarm of robots. *Algorithmica*, 46:193–221, 2006. doi:10.1007/s00453-006-1206-1.

[2] Esther M. Arkin, Michael A. Bender, and Dongdong Ge. Improved approximation algorithms for the freeze-tag problem. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '03, page 295–303, New York, NY, USA, 2003. Association for Computing Machinery. doi:10.1145/777412.777465.

[3] Nicolas Bonichon, Arnaud Casteigts, Cyril Gavoille, and Nicolas Hanusse. Freeze-tag in L_1 has wake-up time five, 2024. arXiv:2402.03258.

[4] Nicolas Bonichon, Cyril Gavoille, Nicolas Hanusse, and Saeed Odak. Euclidean freeze-tag problem on plane. In *36th Canadian Conference on Computational Geometry (CCCG)*, pages 199–205, 2024.

[5] Judith Brecklinghaus and Stefan Hougardy. The approximation ratio of the greedy algorithm for the metric traveling salesman problem. *Operations Research Letters*, 43(3):259–261, 2015. doi:10.1016/j.orl.2015.02.009.

[6] Robert Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, 1943. URL: http://www.jstor.org/stable/2235930.

[7] Ding-Zhu Du and Frank Kwang-Ming Hwang. *Combinatorial Group Testing and Its Applications*. World Scientific, 2nd edition, 2000. doi:10.1142/4252.

[8] Jens Niklas Eberhardt, Nikolas Peter Breuckmann, and Christiane Sigrid Eberhardt. Multi-stage group testing improves efficiency of large-scale COVID-19 screening. *Journal of Clinical Virology*, 128:104382, 2020. doi:10.1016/j.jcv.2020.104382.

[9] David Eppstein, Michael T Goodrich, and Daniel S Hirschberg. Improved combinatorial group testing algorithms for real-world problem sizes. *SIAM Journal on Computing*, 36(5):1360–1375, 2007. doi:10.1137/050631847.

[10] Michael T Goodrich and Daniel S Hirschberg. Improved adaptive group testing algorithms with applications to multiple access channels and dead sensor diagnosis. *Journal of Combinatorial Optimization*, 15:95–121, 2008. doi:10.1007/s10878-007-9087-z.

[11] Mikael Hammar, Bengt J Nilsson, and Mia Persson. The online freeze-tag problem. In *LATIN 2006: Theoretical Informatics: 7th Latin American Symposium, Valdivia, Chile, March 20-24, 2006. Proceedings 7*, pages 569–579. Springer, 2006. doi:10.1007/11682462_53.

[12] Guangjie Han, Huihui Xu, Trung Q Duong, Jinfang Jiang, and Takahiro Hara. Localization algorithms of wireless sensor networks: a survey. *Telecommunication Systems*, 52:2419–2436, 2013. doi:10.1007/s11235-011-9564-7.

[13] Stefan Hougardy and Mirko Wilde. On the nearest neighbor rule for the metric traveling salesman problem. *Discrete Applied Mathematics*, 195:101–103, 2015. 12th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW 2013). doi:10.1016/j.dam.2014.03.012.

[14] Jyh-How Huang, Saqib Amjad, and Shivakant Mishra. CenWits: a sensor-based loosely coupled search and rescue system using witnesses. In *3rd ACM International Conference on Embedded Networked Sensor Systems*, SenSys '05, page 180–191, 2005. doi:10.1145/1098918.1098938.

[15] Richard Kershner. The number of circles covering a set. *American Journal of Mathematics*, 61(3):665–671, 1939. doi:10.2307/2371320.

[16] E.B. Martinson and F. Dellaert. Marco Polo localization. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 2, pages 1960–1965, 2003. doi:10.1109/ROBOT.2003.1241881.

[17] Lehilton Lelis Chaves Pedrosa and Lucas de Oliveira Silva. Freeze-tag is NP-hard in 3d with L_1 distance. *Procedia Computer Science*, 223:360–366, 2023. doi:10.1016/j.procs.2023.08.248.

[18] Daniel J. Rosenkrantz, Richard E. Stearns, and Philip M. Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, 6(3):563–581, 1977. doi:10.1137/0206041.

[19] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, Dec 1997. doi:10.1023/A:1008202821328.

[20] Wikipedia contributors. Marco Polo (game), 2025. URL: https://en.wikipedia.org/wiki/Marco_Polo_(game).

[21] Wikipedia contributors. Tag (game) – freeze tag, 2025. URL: https://en.wikipedia.org/wiki/Tag_(game)#Freeze_tag.

A Additional Related Work

As mentioned in the introduction, the Marco Polo problem falls into a rich area of study known as ***localization algorithms***; see, e.g., the survey by Han, Xu, Duong, Jiang, and Hara [12]. Our approach differs from the approaches used in this prior work, however, in that we are interested in strictly combinatorial strategies, where all we learn is a single in-or-out result from each probe, rather than, say, range and/or directional results, such as in the work by Martinson and Dellaert [16].

The Marco Polo problem is related to combinatorial group testing, see, e.g., [7, 9, 10], which was originally directed at identifying WWII soldiers with syphilis [6] and was recently applied to COVID-19 testing [8]. In this problem, one is given a set of n items, at most d of which are "defective." Subsets of the items (such as blood samples) can be pooled and tested as a group, such that if one of the items in the pool is defective, then the test for the pool will be positive. Tests can be organized to efficiently identify the defective items based on the outcomes of the tests. The Marco Polo problem differs from combinatorial group testing, however, in that the search space for the Marco Polo problem is a geometric region and tests must be connected geometric shapes (i.e., radius-d balls using the Euclidean metric), whereas the search space in combinatorial group testing is defined by a discrete set of n items and tests can be arbitrary subsets of these items.

There is also some work on SAR algorithms that use call-and-response protocols, such as the CenWits system by Huang, Amjad, and Mishra [14], which uses RF-based sensors for the search and rescue of people, such as hikers, who are carrying mobile wireless communication devices in wilderness areas.

In the context of computational geometry, the Marco Polo problem is somewhat related to the Freeze-Tag problem [1–4, 11, 17], which involves "waking up" a collection for moving robots that are initially at given points in the plane via a strategy motivated from the children's game, "Freeze Tag" [21].

We stress that we are interested in solutions to the Marco Polo problem that are adaptive, where the i-th probe can depend on the results of the probes that came earlier. A non-adaptive solution to the Marco Polo problem would be related to a constructive solution to a classic disk covering problem, which asks for the minimum number of disks of radius $\varepsilon > 0$ that can cover a region in the plane [15].

B Supplemental Pseudocode

Pseudocode bounding the distance traveled by the search point, Δ, is provided in Figure 9, and is implemented in https://github.com/ofekih/DroneSearching/blob/main/src/algorithm_utils.py.

Algorithm 1 Bounding the distance traveled by Δ

Require: A probe placement as a list of tuples (x, y, ρ)
Ensure: Upper bound on the total distance traveled

$d_k \leftarrow 0$
$p_{\text{curr}} \leftarrow (0, 0)$ ▷ *Current position*
$b_{\max} \leftarrow 0$ ▷ *Maximum bound*
for each probe (x, y, ρ_k) in placement **do**
  $(c_x, c_y) \leftarrow p_{\text{curr}}$
  $d_k \leftarrow d_k + \sqrt{(x - c_x)^2 + (y - c_y)^2}$
  if probe is last in placement **then**
    ▷ Skip round trip to final probe center
    $(n_x, n_y) \leftarrow$ first probe in placement
    $d_1 \leftarrow \sqrt{n_x^2 + n_y^2}$
    $d_k \leftarrow d_k - 2d_1\rho_k$ ▷ *Subtract round trip*
  $b_{\max} \leftarrow \max(b_{\max}, \frac{d_k}{1-\rho_k})$
  $p_{\text{curr}} \leftarrow (x, y)$
return $b_{\max}$

Figure 9: Algorithm for bounding the distance traveled by Δ. When the POIs are determined to be in the last probe area, Δ can move directly to the first probe of the next search area, saving distance. If the first probe is at distance d_1 from the origin of the original search area, it is at distance $d_1\rho_k$ from the origin of the new search area. The probe placement within the new search area can be rotated such that the first probe is as close to Δ as possible, saving an entire d_1-length round trip to the origin and back.[8]

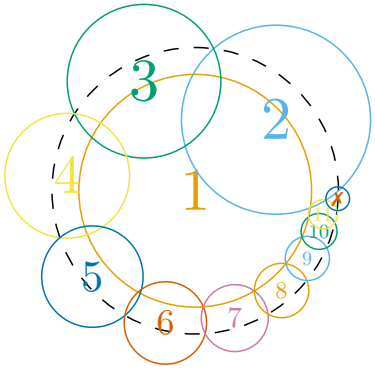
C Omitted Proofs

In this appendix, we provide proofs that were omitted in the body of this paper.

[8] This assumes there is not significant overlap between the last two probes, which holds for all our algorithms.

Lemma 9 (Same as Lemma 1) *A correct progressively shrinking algorithm may require at most* $-\frac{1}{\log \rho_1} \log n + \mathbf{k}_{max} - \mathbf{1}$ *probes to find a POI in a search area of radius* n*.*

Proof. Consider for example the execution of our progressively shrinking Algorithm 6—where the first probe has a proportionality constant $\rho_1 = 0.812$, and, when ignoring constant additive factors as in Equation (3), finds a POI in at most $3.34 \log n$ probes. Algorithm 6 is able to fully cover the search area with its progressively shrinking probes, making it a *correct* progressively shrinking algorithm, and does so by subdividing the search area into 12 progressively shrinking probe regions, as shown in Figure 10.



Algorithm 6

$P(n) < 3.34 \log n$
$\rho_1 = 0.812$
$k_{\max} = 12$

Figure 10: The execution of Algorithm 6 where the POI is at the last region probed by Δ on some layer.

Let us assume that, on the final layer, the POI is in the last search region, as shown in Figure 10, and let us consider three cases:

1. $n = 1$. In this case, we have trivially found the POI without requiring any probes. We also expected to spend at most $3.34 \log 1 = 0$ probes.
2. $n = 12.17$. In this case, we first perform 11 probes, and after they all return negative results, we determine that the POI is in the last region. The size of the last region is $12.17 \times \rho_1^{12} = 1.00$, successfully locating the POI. We spent 11 probes to find the POI, while we expected to spend at most $3.34 \log 12.17 = 12.04$ probes.
3. $n = 1.01$. Similar to the last case, we first perform 11 probes, however this time, the last region is of size $1.01 \times \rho_1^{12} = 0.08$. Not only did we locate the POI within distance 1 of it, we located it to within a much greater precision. Just like the previous case, we spent 11 probes to find the POI, while this time we expected to spend at most only $3.34 \log 1.01 = 0.05$ probes.

All three cases are able to successfully locate the POI, as expected from any correct algorithm. Moreover, the first two cases are able to locate the POI using at most the maximum number of probes expected by $P(n)$. Only the third case, which locates the POI to excess precision, requires more probes, specifically 11 more probes, than expected. Our result follows from the fact that this inefficiency can only occur on the last layer of the algorithm, and that the maximum number of probes we perform on each layer is $k_{\max} - 1$. □

If this added constant is of concern, it can be reduced by increasing the size of all probes to be at least 1. Such optimizations are outside the scope of this paper, however.

It is worth pointing out that Algorithms 1 and 2, which are not progressively shrinking, are able to express this added constant by using $\lceil \log n \rceil$ instead of $\log n$ in the bound for $P(n)$. This is because their value of $k_{\max} - 1$ exactly matches their $\log n$ coefficients of 6 and 5, respectively.

Lemma 10 (Same as Lemma 5) *A circle with radius* r *can be covered by an* L*-layer lattice of hexagons with side length* $s = \frac{2r}{3L-2}$*.*

Proof. Let us consider the case of a hexagonal lattice with an even number of layers, as in Figure 8. The closest point from the center of the lattice to its boundary is obtained by moving diagonally along the center hexagons maximal diameter. Consider the distance, r, to that point. Every even-numbered layer of hexagons contributes one side length s, while every odd numbered layer contributes its maximal diameter $2s$, except for the first layer which contributes s. Thus, the total circumradius $r = \frac{3L}{2}s - s = (\frac{3L}{2} - 1)s$.[9] □

Theorem 11 (Same as Theorem 8) *The total number of probes* (P_{tot}) *and the total distance traveled by* Δ (D_{tot}) *during the memoryless search algorithm for all* k *POIs is at most:*

$$P_{tot} \leq c\lceil \log n \rceil + (c+1)(k-1)\lceil \log \overline{e} \rceil,$$
$$D_{tot} \leq dn + 2dE,$$

where $E < OPT(\lceil \log k \rceil + 1)$*,* $\overline{e} = \frac{E}{k-1}$*, and* OPT *is the optimal tour length for the traveling salesperson problem (TSP) on the* k *POIs.*

Proof. We re-iterate the steps of our stateless below:

1. Find an arbitrary POI using $\mathcal{A}(n)$.
2. Shut off the tracking device of the found POI.
3. Without moving Δ, re-probe the area at radius 2, 4, 8, etc., until a probe returns a positive result (i.e., another POI is found).
4. Invoke $\mathcal{A}$ using this new radius to find another POI.
5. Repeat steps 2 to 4 until all POIs are found.

Other than step 1, the performance of the search strategy depends by the relative positions of the POIs. Let the first POI found be POI 0, and the second POI found in step 4 be POI 1. Let the distance between the two POIs be denoted as e_1, and the distance between POI $i-1$ and POI i be denoted as e_i, for $i \in \{1, 2, \ldots, k\}$. The total number of probes in step 3 required to find a large enough search area containing

[9] This bound is tight for lattices with an even number of layers, and only improves for lattices with an odd number of layers.

POI i is $\lceil \log e_i \rceil$, where the size of the search area is $2^{\lceil \log e_i \rceil}$. Finding the POI within this search area (step 4) will take $P(2^{\lceil \log e_i \rceil}) = c\lceil \log 2^{\lceil \log e_i \rceil} \rceil = c\lceil \log e_i \rceil$ probes. Adding on the initial probe to find POI 0, the total number of probes required to find all POIs is at most:

$$P_{\text{tot}} \leq c\lceil \log n \rceil + \sum_{i=1}^{k-1} (c+1)\lceil \log e_i \rceil.$$

Let $E = \sum_{i=1}^{k-1} e_i$, and $\overline{e} = \frac{E}{k-1}$. At each step of the algorithm, we find a POI that is within a factor of two of the closest still-undiscovered POI to the last found POI. This algorithm will therefore perform at worst a factor of two approximation of the nearest neighbor tour for the traveling salesperson problem (TSP), e.g., see [5, 13]. In Euclidean space, the (greedy) nearest neighbor tour of n salespeople performs at worst a factor of $\frac{1}{2}(\lceil \log n \rceil + 1)$ of the optimal TSP tour length (OPT), see [18]. Applying to our case, $E < \text{OPT}(\lceil \log k \rceil + 1)$. Since $\log e$ is a concave function, it follows from Jensen's inequality that $\sum_{i=1}^{k-1} \lceil \log e_i \rceil \leq (k-1)\lceil \log \overline{e} \rceil$, where $\overline{e}$ is the average value of e, and we obtain our desired result.

Regarding the total distance traveled, note that Δ only moves during steps 1 and 4. For the first POI, Δ travels $D(n) \leq dn$. Using similar reasoning as above, for the i-th POI, Δ travels $D(2^{\lceil \log e_i \rceil}) \leq d(2e_i)$. Overall, the total distance traveled is at most: $dn + 2d\sum_{i=1}^{k-1} e_i = dn + 2dE$, and our result follows. □

D Algorithm Assumptions

In this section we justify some assumptions regarding the possible probes and locations of the POIs that we make in the main body of the paper. Namely, we assume that:

- The maximum allowed probe distance, d, is n.
- There may be multiple POIs, either within the initial search region or slightly outside of it.

More specifically, we show how, by relaxing these assumptions, a relatively simple solution is able to solve the problem using an optimal number of probes.

Simple Case: Exactly One POI at Full Distance. Let us begin with a simple (but admittedly unrealistic) scenario, where there is exactly one POI within distance n of the origin and we can perform arbitrarily large probes. In this case, where we allow unbounded probe distances, we can find the sole POI using two binary searches, one in each dimension. We place Δ at a very large distance from the origin, such that the intersection for the probe with the original search region is nearly a straight line, and perform one binary search using $\lceil \log n \rceil + 1$ probes to find a region of width at most 1 in the first dimension. We then perform a similar binary search placing Δ in an orthogonal direction to find the POI in the second dimension. We are left with a region of side length 1, which is fully contained within a single, final, radius-1 probe. This solution thus uses $2\lceil \log n \rceil + \mathcal{O}(1)$ probes, but it relies on Δ traveling very far away and at high altitude and the tracking device to have very strong signal strength, which are not reasonable assumptions. This can be improved somewhat by restricting the probe distance to n, as we explore in the next section.

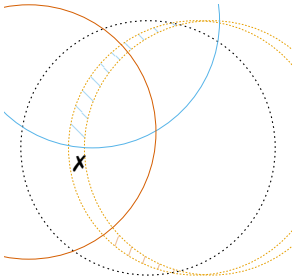


Figure 11: A search for one POI (✗), first reducing the horizontal dimension to a width of 1 (in the dashed orange probes), then searching along the remaining arc (with the solid blue and orange probes).

A Slight Improvement to a Bad Algorithm. While the previous algorithm allows for unbounded probe distances, we now show how to achieve a similar result while restricting the probe size to be at most n. In this case, we reduce the horizontal dimension this time first to a 1-width arc, and then cut the length of the arc in half for each subsequent probe. Note that the arc may be up to πn units long, which is greater than the $2n$ original region diameter, resulting in up to one more probe. This probe can be removed by reducing the remaining *area* by two per probe in the first dimension rather than the arc width. See Figure 11.

Note that this algorithm may require Δ to travel outside of our original search region. This can be avoided with a final algorithm. We can place Δ at the center of the original search region and perform a binary search by changing the radius of the probes, resulting in at most a width-1 shell of the original search region, requiring at most $\lceil \log n \rceil$ probes.[10] This shell of outer-radius r is $2\pi r$ units long, where r can be up to n. We can reduce this shell using a similar binary search, by moving Δ along the outer edge of the shell, performing probes with radius r. See Figure 12 for an example of this algorithm. Note that the first of these probes will only reduce one third of the shell, rather than a factor of 2, resulting in up to one more probe overall. We can then perform probes to reduce this shell to a width-1 square, which could be performed in $\lceil \log n \rceil + 2$ probes.

E Darting Non-Monotonic Algorithms

In this section we discuss the darting algorithms, which are non-monotonic and place many more probes at each recursive level by first placing several carefully-placed probes and then greedily adding more to fill in smaller and smaller gaps, albeit at the expense of darting from side to side in the search space to do so. That is, the approach for our darting algorithms is as follows: after an initial placement of several probes, we determine if there are any uncovered internal areas. If so, we find the largest such area and place a probe

[10] Note that reducing the remaining *area* by a factor of two at this stage would reduce a constant number of probes.

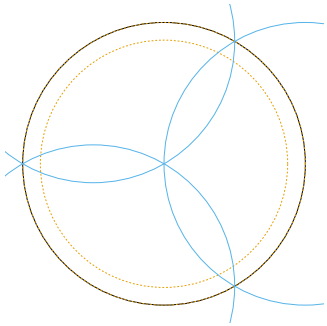


Figure 12: A depiction of the final "simple" search strategy where it is known that there is only one POI within the search region. In this final strategy, not only is the probe distance limited to n, but Δ is also restricted to the original search region.

such that it intersects two points on the area's convex hull. If there are multiple possible placements, we repeatedly choose the one that reduces the remaining area the most in a greedy fashion. See Figure 13. Subsequent probes are placed in the same manner, until either the entire search area is covered, or the probes become sufficiently small as to be unable to cover the remaining area.

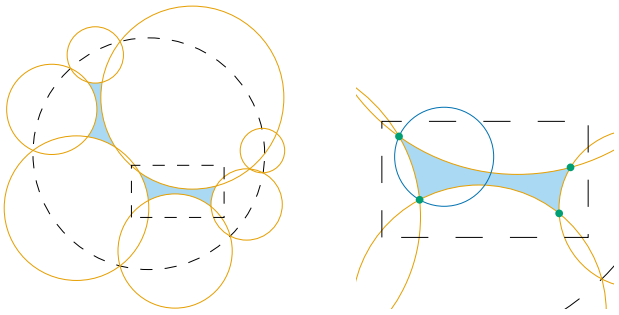


Figure 13: A method to greedily add an arbitrary amount of probes to any initial placement of probes. On the left, Algorithm 4 given an insufficient ρ_1 value fails to cover the search area, leaving some uncovered internal area (in blue). On the right, the largest uncovered internal area's convex hull (in green) is identified, and a probe is placed such that it intersects two points on the hull.

We note that the final probe of Algorithm 4 is often the least efficient; see Figure 13 (left) and Figure 4. If we run Algorithm 4, remove the final probe, and then apply the greedy method described above to place the remaining probes, we obtain Algorithm 7, which is able to significantly reduce the number of probes required to find a POI to $P(n) < 2.93 \log n$.

Exploiting this approach further, however, strains our ability to reason about regions that are uncovered after performing many probes; hence, for an even further improved algorithm, Algorithm 8, its probe sequence for each recursive level is determined using computer-assisted proof. More specifically, the placement of the first six probes is determined by a differential evolution algorithm, which is a type of genetic algorithm that optimizes a function by iteratively improving a population of candidate solutions, see [19]. Next, we fill in the gaps according to the aforementioned greedy method. The resulting Algorithm 8 is able to achieve our best probe results, with $P(n) < 2.53 \log n$, albeit with a very large value for $D(n)$. See Figure 7.

F Experiments

We implemented our 8 algorithms and tested them. The data were obtained by placing a POI at a random location, determined from a uniformly random angle and a random distance from the center of the search area. When the POI was in the last probe's search area, the last probe was not executed, and Δ proceeded directly to the first probe of the next search area. Each algorithm was executed 40 million times, where $n = 2^{20}$, and normalized by dividing by either $\lceil \log n \rceil$ or by n. Our code is publicly available at `https://github.com/ofekih/DroneSearching`. Both Algorithms 1 and 2, despite having a poor worst-case probe complexity, perform well in practice, finding the POI using fewer probes on average than Algorithms 3–6. Our computer-assisted algorithms (7 and 8), however, outperform them. Interestingly, the progressively shrinking algorithms all have a non-zero variability in their probe counts. See Figure 14. While Algorithms 1 and 2 perform well in terms of average distance traveled, they are outperformed by Algorithms 5 and 6, which each have the best distance-traveled guarantees. See Figure 15. Overall, the best methods are Algorithm 8 if number of probes is a priority, Algorithm 6 if distance traveled is a priority, and Algorithm 2 for a good balance.

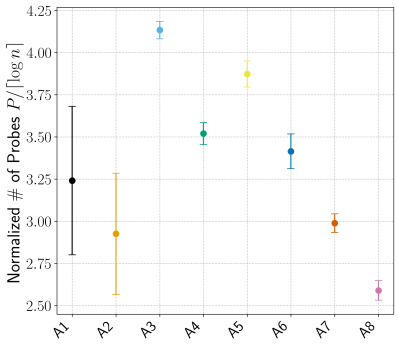


Figure 14: Simulation results for $P/\lceil \log n \rceil$. Error bars represent one standard deviation from the mean.

Number of Probes Made. We make two interesting observations regarding the number of probes made, P, for our progressive probe algorithms.

Observation 1 *The progressive probe algorithms, Algorithms 3–8, exhibit a non-zero variance in P experimentally.*

And perhaps more surprisingly, in Table 1 we observe:

Observation 2 *Progressive probe algorithms appear to perform more probes than their theoretical upper bounds.*

Category	Alg. #	Probes ($P/\log n$) Min	Avg	Max	Bound	Total Distance (D/n) Min	Avg	Max	Bound	Responses ($R/\log n$) Avg	Max	Bound
Hexagonal	Alg. 1	**1.00**	3.24	5.70	6.00	**0.00**	3.35	10.39	10.39	**0.89**	**1.00**	**1.00**
	Alg. 2	**1.00**	2.93	4.80	5.00	**0.00**	2.65	8.81	8.81	1.11	1.45	2.00
Chord-Based	Alg. 3	3.85	4.13	4.25	4.08	4.69	5.46	6.56	6.95	1.99	2.40	4.08
	Alg. 4	3.10	3.52	3.70	3.54	4.30	5.38	9.00	9.31	1.94	2.50	3.54
Monotonic	Alg. 5	3.55	3.87	4.15	3.83	**0.00**	**1.92**	6.72	6.72	2.49	3.85	3.83
	Alg. 6	3.25	3.41	3.85	3.34	**0.00**	1.96	**6.01**	**6.02**	1.96	3.35	3.34
Darting	Alg. 7	2.90	2.99	3.65	2.93	3.86	5.97	25.74	25.80	1.39	2.15	2.93
	Alg. 8	2.55	**2.59**	**3.20**	**2.53**	2.44	4.05	42.58	45.40	1.31	1.85	2.53

Table 1: A numerical comparison of simulation results for our 8 algorithms on three normalized performance metrics, namely the number of probes made (P), the total distance traveled by Δ (D), and the number of POI responses (R). The best values are highlighted in bold. The category names used are crude abbreviations; see the main paper for their proper names.

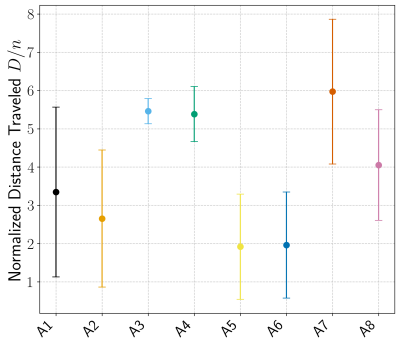


Figure 15: Simulation results for D/n, where $n = 2^{20}$.

Category	Alg. #	Probes ($P/\log n$) Max	True Bound
Hexagonal	Alg. 1	5.70	6.00
	Alg. 2	4.80	5.00
Chord-Based	Alg. 3	4.25	4.28
	Alg. 4	3.70	3.9
Monotonic	Alg. 5	4.15	4.18
	Alg. 6	3.85	3.89
Darting	Alg. 7	3.65	4.18
	Alg. 8	3.20	4.13

Table 2: A table comparing the maximum number of probes observed in our experiments with the true upper bounds for each algorithm when $n = 2^{20}$.

Both observations are primarily explained by the last recursive layer in each of our algorithms. In the proof of Lemma 1 in Section C, we show how the number of probes required in the last layer depends significantly on the size of the last layer as well as the placement of the POI within it. Specifically, it can vary by up to the maximum number of probes performed per layer of the algorithm, $k_{\max} - 1$, and thus the true maximum number of probes performed can be up to $k_{\max} - 1$ more than our simplified bounds as shown in Table 1 (see Lemma 1). To obtain more precise upper bounds for any specific n, we can expect our normalized number of probes made, $P(n)/\log n$, to be increased by up to $\frac{k_{\max}-1}{\log n}$. As expected, these adjusted bounds now hold for all algorithms. See Table 2. As n increases, the $\log n$ coefficient in $P(n)$ dominates this constant number of extra probes, so our theoretical upper bounds will hold.

Total Distance Traveled by the Search Point. As expected, both our higher-count monotonic-path (HM) algorithms, Algorithms 5 and 6, minimize the total distance traveled by Δ. Algorithms 1–3 also have monotonic counter-clockwise paths, but their less efficient probe sequences result in worse average distance traveled. See Table 1. The hexagonal and HM algorithms each start with a central probe,[11] and consequently have a minimum of 0 distance traveled. While all algorithms experienced an instance where Δ traveled nearly as much as their worst-case bound, reassuringly, most algorithms performed significantly better on average, with the hexagonal and HM algorithms performing ~3 times better, and the darting non-monotonic-path algorithms, Algorithms 7 and 8, performing ~4 and ~11 times better, respectively. They are still likely not best suited for time-critical rescue operations since they travel 2-3 times more than the HM algorithms on average.

Number of POI Responses. Finally, we compare the number of POI responses, R, for our algorithms, to see which are best suited for the case where the POI has battery constraints. We find that, Algorithm 1 performs by far the best, with not only the fewest responses across the board, with its maximum number of responses being lower than the average of any other algorithm, but also with the smallest standard deviation. See Figure 16. This result is not surprising, because in Corollary 7 (3), we learn that Algorithm 1 is part of a larger family of response-efficient algorithms. The next best algorithm is Algorithm 2, also partly based on a hexagonal grid, and then the darting algorithms, Algorithms 7 and 8, which perform only a small number of probes in general.

[11] Recall that the hexagonal algorithms can be modified to start with the first probe instead of ending with it.

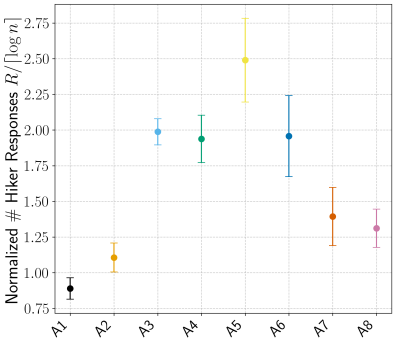


Figure 16: Simulation results for $R/\log n$