TREASURE HUNTING

CASEY O'BRIEN (CMOBRIEN@MIT.EDU)

Our goal is to solve the collaborative search on the plane problem. In this problem, there are k identical agents, initially located at the origin, and a treasure located Manhattan distance D from the origin. Their goal is to locate the treasure as efficiently as possible, both in terms of D and k.

The agents all move synchronously. In a single time step, an agent can take one step to the north, east, south, or west, or stay in the same location. Our goal is to develop an algorithm for the agents to locate the treasure in the optimal time, $O(D+D^2/k)$, when the agents have only weak capabilities. Specifically, our agents will only have constant memory, and very limited communcation abilities. The only communication allowed will be *loneliness detection*, which means that when an agent is in a cell, it knows whether or not it is alone. Additionally, the agents will be able to sense whether their current cell is the origin.

1. Getting Started

To begin, we will make three assumptions about the agent's capabilities which do not fit into our model. In this section, we will describe an algorithm, Rectangle-Search which will allow the agents to locate the treasure under these assumptions. Then, in section 2, we will show how we can lift the assumptions. In Section 3, we will analyze the runtime of the algorithm.

1.1. Assumptions.

We make the following assumptions.

Assumption 1. *The agents are broken into teams of five, and each agent knows its role within its team.*

Assumption 2. *Each agent knows whether or not it belongs to the first team.*

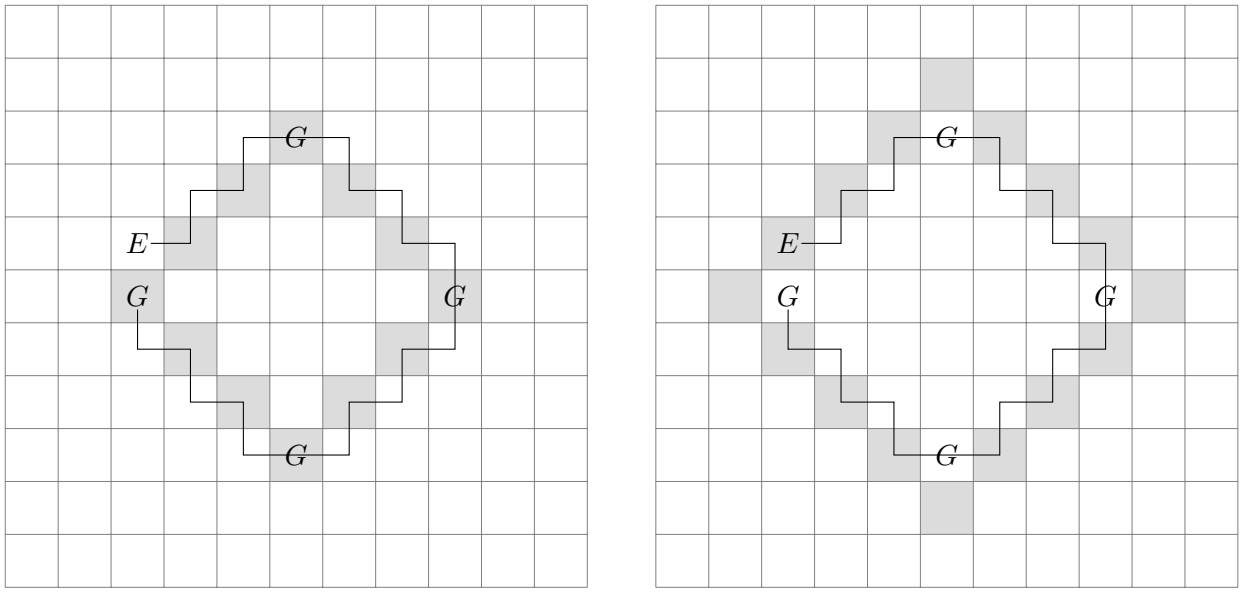
Assumption 3. *All the agents begin at the origin. At time* $12i$*, team* i *leaves the origin and begins to execute the algorithm.*

1.2. Exploring Levels.

Define *level* d to be the set of all cells which are exactly d steps from the origin. The general idea behind the algorithm is that search teams will concurrently search different levels, starting from level 0 and making their way outwards. Once a team finishes searching a level, it will move outwards again and explored the next unexplored level.

In order for this algorithm to work, the agents will have coordinate with each other very carefully. To do this, each agent of a search team is assigned one of five roles: the explorer, or the north, east, south, or west guide.

Say that a search team is going to explore level d. The guides will position themselves on their corresponding axis exactly d steps from the origin, and the explorer positions himself one cell above the west guide. Now, in order to explore level d, the explorer begins by alternating taking steps to the east and north, until it reaches the north guide, at which point it begins alternating taking steps to the east and south until it reaches the east guide, and so on. After $8d-1$ steps, the agent will have reached the west guide, and all cells at level d will have been explored. Figure 1.1(a) shows an example of the path of the explorer for $d=3$, where the cells at level 3 are highlighted.



(A) (B)

FIGURE 1.1. TEST

In fact, we note that all cells at level $d+1$ will have also been explored, except for those lying exactly one step further from the origin than the guides. To demonstrate this point, Figure 1.1(b) shows the path of an explorer who is exploring level 3, with the level 4 cells highlighted. Soon we will see that these cells along the axes will be explored by the guides, and so it is unnecessary to have any team explore level $d+1$. As a result, we only need to designate a search team to explore every other level.

1.3. **Progression.** Now we need to make sure that the guides use only constant memory. As described above, the guides need to move d steps from the origin before the search at level d can begin. In order to do this using only a constant amount of memory, the guides will make use of guides from other search teams.

The algorithm begins with the first team exploring level 2. At time $t = 0$, the explorer takes a step along the west axis. At time $t = 1$, the explorer takes another step, and all the guides begin walking out along their respective axes. Once the explorer reaches the cell $(-2, 0)$, it takes a step upwards. After this step, the team is in position to begin exploration of level 2, and the explorer proceeds to search the level as described above.

While the first team is exploring level 2, at $t = 12$ the second team begins to execute the algorithm. As before, the explorer leads the west guide out along the west axis, and all other guides walk out in sync with the west guide. These agents are not actually aware that they are the second team (they only know that they are not the first team), so we need a strategy to let the agents know when to stop walking outwards and begin exploring a level.

Once the agents leave the origin, they walk until they reach an empty cell an even distance from the origin. This corresponds to the inner-most even level which has yet to be explored. So, once the team reaches this point, they can begin exploration of that level.

All subsequent teams then follow the same pattern. Since we have $\lfloor k/5 \rfloor$ teams, this will result in the first roughly $\lfloor 2k/5 \rfloor$ levels being explored. However, if the treasure still has not been located, we need a way for the agents to continue searching.

To handle this, we want to make the teams move outwards and find a new level to search once they have finished searching their current level. Consider the first team, right after it finishes searching level 2. Once the team finishes searching, the entire team can move outwards as they had before, again stopping once they find an unoccupied even cell.

Consider a new team which leaves the origin after the inner-most teams have already moved outwards to start exploring new levels. We wouldn't want this new team to mistakenly search one of the inner levels that has already been searched. As a result, we have the new team walk until it locates another team. After that point, it settles in to search the first even level that no team is currently exploring.

The issue with having teams move outwards after the explorer finishes is that there is no way for the north, east, and south guides to know exactly when the explorer has gotten all the way back to the west guide. However, we note that this isn't actually necessary. The north guide can begin its move outwards as soon as the explorer passes by it (and the same for the east and south guides). Since the explorer and west guide will begin their move outwards after the other guides, we know that those guides will be in place by the time the explorer reaches them when searching the new level.

It remains to explain how the guides know when the explorer passes them. For now, lets just consider the north guide. The east and south guides will be treated symmetrically (although we will see that the west guide is different). There are three situations in which one of these guides might find another agent in their cell while they are waiting for their explorer:

(1) Their explorer is passing them.
(2) The north guide from a new team is passing them.
(3) The north guide from a team moving outwards to explore a different level is passing them.

Recall that north guides are emitted from the origin every 12 steps. We also know that there will be a large gap between two teams moving outwards to explore a new level, because even if they started exploring at the same time, the inner team would finish many steps sooner. We can utilize these facts to make the north guide "recognize" its explorer. If a north guide ever experiences three consecutive time steps where it is not alone, at least one of those time steps must have consisted of the explorer passing.

To utilize this fact, whenever the explorer reaches the north guide, it waits there for four time steps (in fact, only three are necessary, but we want to maintain that the explorer takes an even number of steps before it reaches the west guide. We'll see later why this is important). This way, if the north guide is ever not alone for at least four time steps, it knows that its explorer has visited.

The north guide should not begin its move outwards until it senses it is alone, because if two north guides moved outwards at the same time, they would never stop, because they would never be alone. However, the north guide can only possibly be not alone for at most six time steps (four from the explorer, and one from each type of north guide), so he will still make his move outwards long before the explorer begins its outward move.

Now, we must consider the west guide. When the explorer reaches the west guide again, we want them to move outwards together, with the explorer one step ahead of the west guide. If we use the same strategy as we did for the other guides, we could end up in a situation where the explorer and west guide are separated by another set of explorers and west guides. The result would be two explorers, followed by two west guides, which would result in two explorers exploring the same level.

The solve this problem, we will ensure that our algorithm maintains the following invariant.

Invariant 1. *During any odd time step, all west guides have an even x coordinate.*

Recall that at time $12i$, the explorer from team i leaves the origin, and at time $12i + 1$, the guides from team i leave the origin. So, we can clearly see that this invariant holds while the west guides are making their initial move out along the west axis. Any guide waiting for its explorer to search a level is positioned on an even cell, so the invariant is clearly satisfied during the time.

It remains to show that a guide moving outwards after its team has already explored a level follows this invariant. We have yet to describe how the west guide and explorer initiate their move outwards after exploring a level.

The tricky thing here is that it is important that the west guide knows exactly when the explorer returns. The north guide, for example, can't tell the difference between an explorer passing directly followed by another north guide, and a north guide passing directly followed by the explorer.

To rememdy this, we slightly alter the behavior of the west guide. Say that a west guide has just reached its position and the explorer has just left. Then, the explorer and west guide from another team pass by the west guide. At that point, the original west guide should take a step south, so that it sits just below the west axis.

This won't change anything for any new teams moving out into position, because after the first team is deployed there will always be at least one west guide sitting on the west axis, and so teams will still start searching at the appropriate new levels. Now, the west guide knows when its explorer returns because no other agent could be at that position directly below the west axis.

We have already argued that the explorer takes an even number of steps to get back to the west axis. Since the explorer is one step ahead of the west guide walking out along the west axis, it starts its exploration on an even time step. Thus, we know that the time step in which the explorer reaches its west guide below the axis is an odd time step.

In the next time step, we have the explorer move one step to the left, so that it sits next to its guide below the axis on an even time step. Then, together, they take a step up so that they are on the west axis (resulting in an odd time step).

Due to the invariant, if the guide is not alone at this point, it must be in the cell with another guide. Since explorers and guides always walk one after another, the explorer must also find itself with another explorer. Thus, they can wait two time steps for that team to pass, and then they can follow. Figure 1.2 shows an example where a team finishing its search enounters another team and needs to wait for it to pass.

It's possible that there are two teams walking in a row, in which case they may need to wait four time steps. Either way, they always wait an even number of steps before they begin moving, so they preserve the invariant.

If the explorer and guide find themselves alone when they return to the west axis, they know that their teammate must be alone as well, so they can begin their westward walk immediately.

With this argument, we can see that the invariant is preserved across all possible scenarios.

At this point, we have an algorithm which will locate the treasure under the three original assumptions. We will hold off on analyzing the runtime of the algorithm until we have described the entire algorithm. In the following section, we describe how to modify the algorithm so it works even when we lift the assumptions.

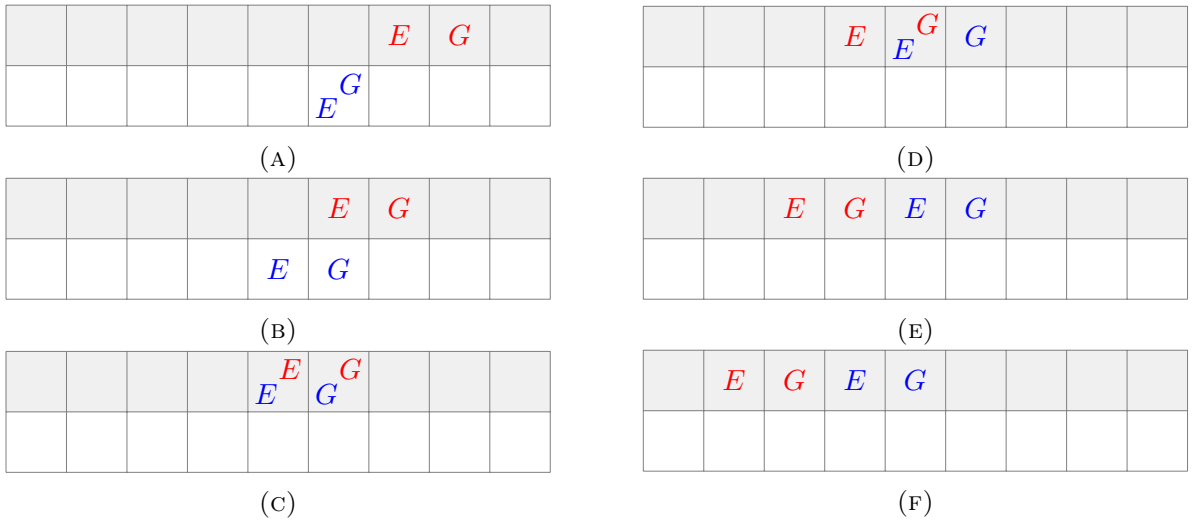


FIGURE 1.2

2. Lifting Assumptions

Now, we describe how to lift our assumptions. First, we describe an addition to the algorithm which allow us to lift assumptions 1 and 3. We will do this by having the agents perform a routine before they begin executing the algorithm described in the previous section. This routine will be broken into two phases. In phase I, agents will essentially line up to enter phase II. In phase II, the agents will determine their roles for the rest of the algorithm.

First, we introduce a new definition. Define the point $(1, 1)$ to be the *center*. Instead of centering our search around the origin, we will center it around the center (soon we will see why this is necessary). We will redefine level d to be the set of points exactly d steps from the center. Since the center is two steps from the origin, this means that we will need to search all the way through level $D + 2$ in order to ensure that the agents have located the treasure.

2.0.1. *Phase I.* To begin, in the first time step, we have all agents move one step to the left, to the position (-1, 0). Now, each time step, we have the agents move to the left with probability 1/2 (and otherwise stay in the same cell). Once an agent finds itself alone in a cell, it takes a step to the right every time step until it reaches the origin. For reasons we will see later, we only want an agent returning to the origin at most once every other time step. Thus, during this phase, the agents will move only on even time steps. When an agent returns to the origin, they enter phase II.

2.0.2. *Phase II.* Once the agents get back to the origin (which they can recognize, because this is one of their limited abilities), they follow the path outlined in Figure 2.1. Say that an agent takes the first step past the origin and finds itself alone. This will indicate to the agent that it is going to be a north agent. We want the next four agents that pass to know that they are not meant to be north agents. So, that agent will wait until it has been passed by four other agents. For any agent who passes that square and finds itself not alone, it will know it is not a north agent and continue moving along the path.

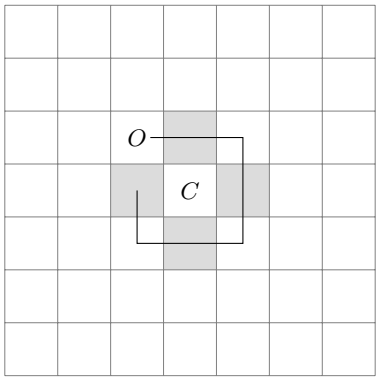


FIGURE 2.1

Once the designated north agent has been passed by four agents, it knows that the rest of its team has passed, and begins to walk north. At this point, the north agent behaves exactly like a north agent emitted from the origin in the original algorithm in Section xx.

The east and south agents are designated the same way. If an agent reaches the cell directly to the right of the center and finds it unoccupied, it knows it will become an east agent and waits for three other agents to pass before moving outwards. If an agent reaches the cell directly south of the center and finds it unoccupied, it knows it will become a south agent and waits for two other agents to pass.

The west agent is slightly different. If an agent reaches the cell to the left of the center and finds it unoccupied, it knows it will become the west agent. However, it waits a single time step before moving outwards. This is because of the explorer.

As soon as an agent reaches the step to the left of the center and finds it occupied, that agent knows it is an explorer, and it takes a step to the left in the next time step. By waiting a single extra time step, the west guide ensures that it does not walk out on top of the explorer, in which case neither could sense if they were in the presence of other agents. Note that because agents only return to the origin every other time step, it is impossible for another agent to encounter a west agent while the agent is waiting that extra time step before moving westwards.

Figure 2.2 shows a sample execution of phase II on five agents. Since phase I involved randomness, we display an execution where we assume that the agents executed phase I in such a way that they end up entering the origin exactly once every other time step.

We are left with a final issue of dealing with the fact that currently, we need the agents to know whether they are the first team to leave the origin. In Figure 2.2, we see that the guides all stop exactly two steps from the center, and the explorer begins searching level 2. However, they must know that they are the first team in order to do that. Recall that agents from every subsequent team walk away from the origin until they find an empty even cell *after* having passed by another agent. Of course, the first team cannot do this because they will never encounter another agent and therefore walk on forever.

To handle this, we introduce another phase to the algorithm above, which we'll call phase X. To tell our agents when to enter phase X, we need to modify the end of phase I slightly. If an agent reaches the origin and finds itself alone, it immediately enters phase X (to be described shortly). If an agent enters the origin and is not alone, it immediately enters phase II.

Phase X is very similar to phase II. The agents will walk along a predetermined course. If they find themselves alone in certain cells, they will stop and wait there until some condition indicates

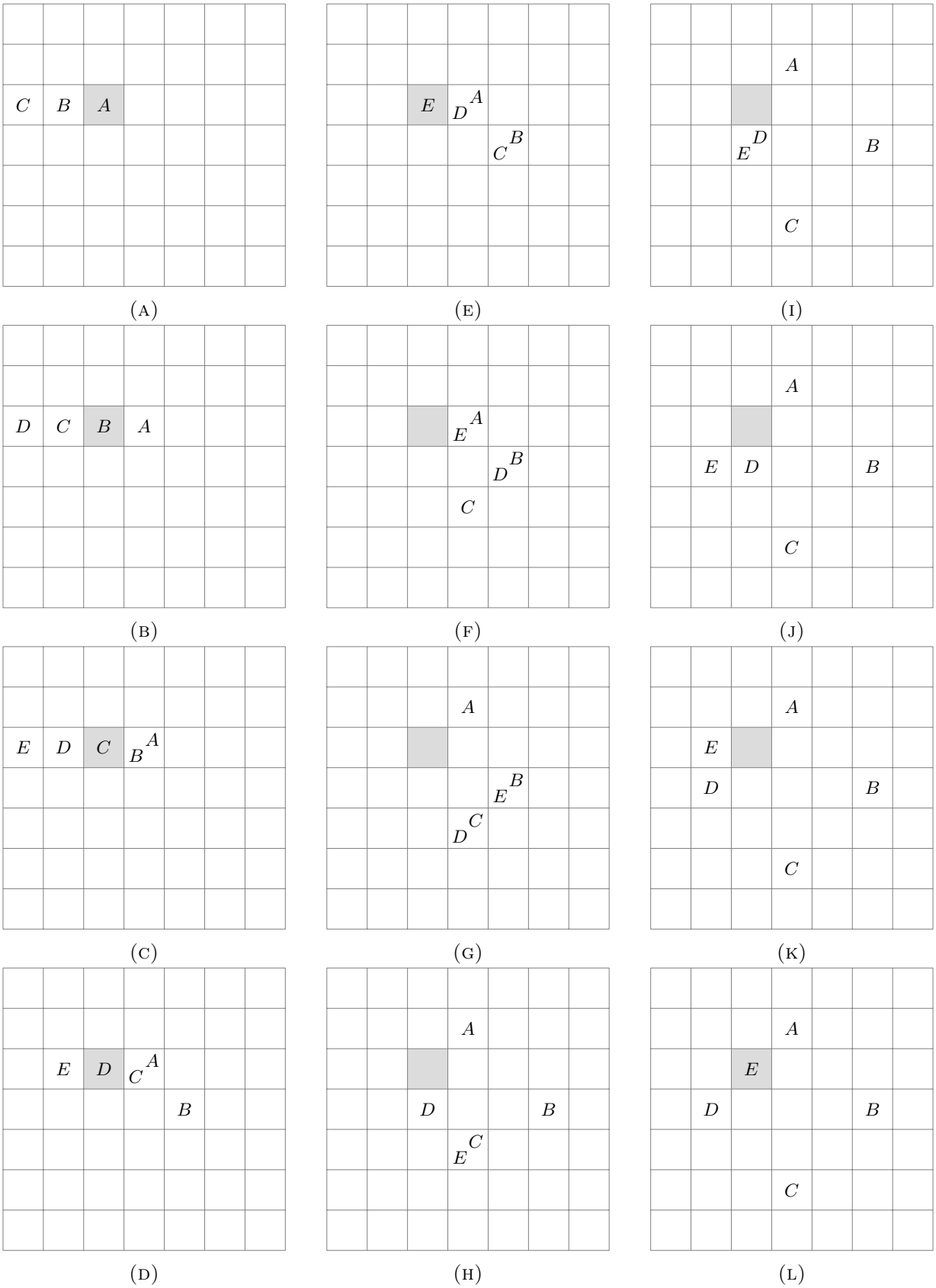


FIGURE 2.2. TEST

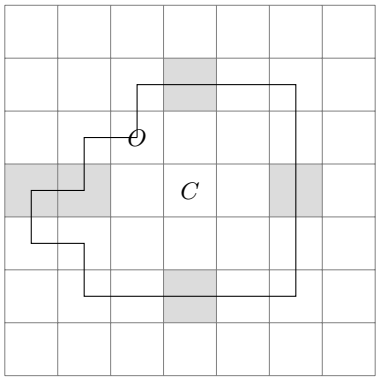


FIGURE 2.3

to them that they can move forward. Figure xx shows that path that agents execute in Phase X. As in Figure xx, the gray cells show the ones in which agents stop in if they find themselves there alone.

Some agents may complete this path (after passing all five gray cells and finding them occupied) and return to the origin. Once a phase X agent returns to the origin, it remains there for the remainder of the algorithm. As soon as at least one phase X agent return to the origin, all other agents that are still in phase I will enter phase II after reaching the origin.

At most 13 agents can ever enter phase X by the time one of those agents returns to the origin and agents start to enter phase II. As we will see, the five agents stopped in the gray spots will later form a search team, so this means that there will be at most 8 agents who return to the origin and never leave again.

Now, when a team enters phase II, it can just assume that it is not the first team. This is because all phase X agents will still be positioned in their gray spots, and so even the first team will pass by them and know that they can search the next unoccupied level.

The problem is once the agents start completing their levels and moving outwards. If there are still agents in phases I and II at that time, we do not want them to re-search early levels after seeing phase X agents. So, we actually just want the phase X agents to be in position until at least one team has passed them.

We remedy this as follows. When the first phase II team begins to move outwards, it will counter agents at level 2. Only phase X agents will ever be stopped there, so it will know the agents are phase X agents. The phase II agents will then wait in the cells with the phase II agents for an extra time step before proceeding.

A phase X agent will never find its cell occupied for more than one time step in a row until this point, because agents can only enter phase X every other time step. As a result, this will signal to the phase X agents that they can abandon their posts and begin searching themselves.

The first team out of phase II will step and begin searching level 4. Once the phase X team has received notification this this has happened, they can move outwards and begin searching level 6. At this point, that team will behave exactly like a normal search team.

Note that all level 1, 2, and 3 cells are either searched by phase II or phase X agents, or lie on an axis and are searched by guides, so there's no need to ever have a team explore these levels. We have to be careful because no agent ever actually explores the center. To remedy this, we just have all the agents move to the center and back to the origin before beginning phase I.

3. Analysis

Theorem 1. Rectangle-Search *locates the treasure in* $O(D+D^2/k+\log k)$ *with high probability.*

Proof. First, we need to analyze the rate at which agents complete Phase I. To do so, we use the following lemma.

Lemma 2. *After* $O(t+\log k)$ *time steps, at least* t *agents have completed phase I with high probability.*

Proof. Still need to work this out exactly. The general idea is that is takes $O(\log k)$ steps for one of the agents to become alone, and after that happens we expect roughly one agent to become alone every time step. Very roughly. □

Next, we assess the time until one of the Phase X agents return to the origin (which marks the point at which Phase I agents start moving into Phase II). As we've stated before, at most 13 agents can enter Phase X before one of them winds up back at the origin. Once the last agent enters phase X, it will take at most a constant number of steps before some phase X agent reaches the origin. Thus, by Lemma xx, we expect this to happen in $O(\log k)$.

Define team 0 to be the team which is formed in *Phase X*, and define team i to be the i^{th} team formed in phase *II*. Next, we will define team j to be the last team which exits *Phase I*. Since the agents move in synchrony, the runtime of the algorithm is equal to the number of steps before team j stops searching.

Lemma 3. *We must have that* $j = O(D)$.

Proof. Even if each team only explored a single level, the $D/2^{th}$ team would end up exploring level D. After $O(D)$ time steps, that team would complete exploration of level D and the algorithm would terminate. Since at most one team can start per time step, this leads to $j = O(D)$. □

Let m be the maximum number of possible teams that can be formed from the k agents. Recall that at least 1 and at most 8 agents will complete *Phase X* and then return to the origin to never leave again. So, we have that $\lfloor (k-8)/5 \rfloor \leq m \leq \lfloor (k-1)/5 \rfloor$, and $m = \Theta(k)$.

We analyze two cases. In the first case, $j = m$. This corresponds to the case where D is large compared to k, so all possible teams are deployed and join the search before the treasure is found. In the second case, $j < m$. This corresponds to the case where D is small compared to k, so the treasure is located before all the teams are deployed.

Lemma 4. *If* $j = m$*, then the algorithm terminates after* $O(D + D^2/k)$ *steps with high probability.*

Proof. This means that D was large enough such that all the agents completed *Phase I* and participated in the search. By Lemma xx, it takes $O(k + \log k) = O(k)$ with high probability before team $k/5$ completes *Phase I*. After that, the team moves out to find the first unexplored level. Say that the first unexplored level it finds is level f_j.

Once there, the team explores that level. Recall that it takes $O(d)$ steps to explore level d. So, in $O(f_j)$ steps, the team explores level f_j. Next, they move out and explore the next unexplored level, which will be level $f_j + 2m$. After exploring that level, they'll move out and explore level $f_j + 4m$. This process can repeat at most $D/2m$ times. So, the total amount of work is given by the following sum.

$$
\begin{aligned}
f_j + \sum_{z=0}^{D/(2m)} O(f_j + z \cdot (2m)) + m \\
&= f_j + O\left(\sum_{z=0}^{D/k} f_j + zk\right) \\
&= f_j + O\left(\sum_{z=0}^{D/k} f_j + k\sum_{z=0}^{D/k} z\right) \\
&= f_j + O((D/k) \cdot f_j + D^2/k)
\end{aligned}
$$

To simplify this further, we claim that $f_j = O(D)$. To see why, we note that if $f_j > D$, then it means that by the time that team j starts its search, some other team is already searching level D. That team will complete its search within $O(D)$ steps of the time when team j begins its search, so team j cannot possibly start a search further than $O(D)$ steps from the origin. With this in mind, the above sum simplifies to

$$= O(D + D^2/k + D^2/k) = O(D + D^2/k)$$

With this, the total time before the algorithm terminates is

$$O(D + D^2/k + k).$$

By Lemma xx, $k = O(D)$, so we have that the overall runtime is given by $O(D + D^2/k)$ with high probability. □

Lemma 5. *If $j < m$, then the algorithm terminates after $O(D + \log k)$ steps with high probability.*

Proof. This means that not all teams completed *Phase I* by the time that some team located the treasure.

We know that team $j + 1$ never completes *Phase I*, because the algorithm terminates before this. By Lemma xx, this team would have completed *Phase I* in $O(j + \log k)$ time steps with high probability. Thus, the algorithm completes in $O(j + \log k)$ with high proability. Then by Lemma xx, the algorithm completes in $O(D + \log k)$ with high probability. □

Combining Lemmas xx and xx, we have that the algorithm terminates in $O(D + D^2/k + \log k)$ with high probability.

□