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. Introduction

We will describe the algorithm Rectangle-Search. It consists of three phases. The first phase is the *Separation Phase.* This is the only probabilistic phase. In this phase, the agents work to become alone in a cell, so that they can then make use of their loneliness detection ability.

The second phase is the *Allocation Phase.* In this phase, the agents divide up into teams of five, and each agent is assigned a role within its team. The final phase is the *Search Phase.* In this phase, the agents actually begin to search the plane for the treasure by collectively searching from the origin outwards.

Since the *Search Phase* comprises the majority of the algorithm, we will describe it first, in Section 2. In Section 3, we will describe the *Separation Phase*, and in Section 4 we will describe the *Allocation Phase.*

2. Search Phase

Since the *Separation* and *Allocation Phases* actually happen before the *Search* phase in *Rectangle-Search*, we will need to make some assumptions about the behavior of the agents entering the *Search Phase* before we can proceed to describe this phase. In later sections, we will see how the early phases actually allow these assumptions to be valid.

2.1. 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. A search team can leave the origin on any even time step at least twelve steps after the previous team left. The second search team must leave the origin before the first search team completes exploring level 2.*

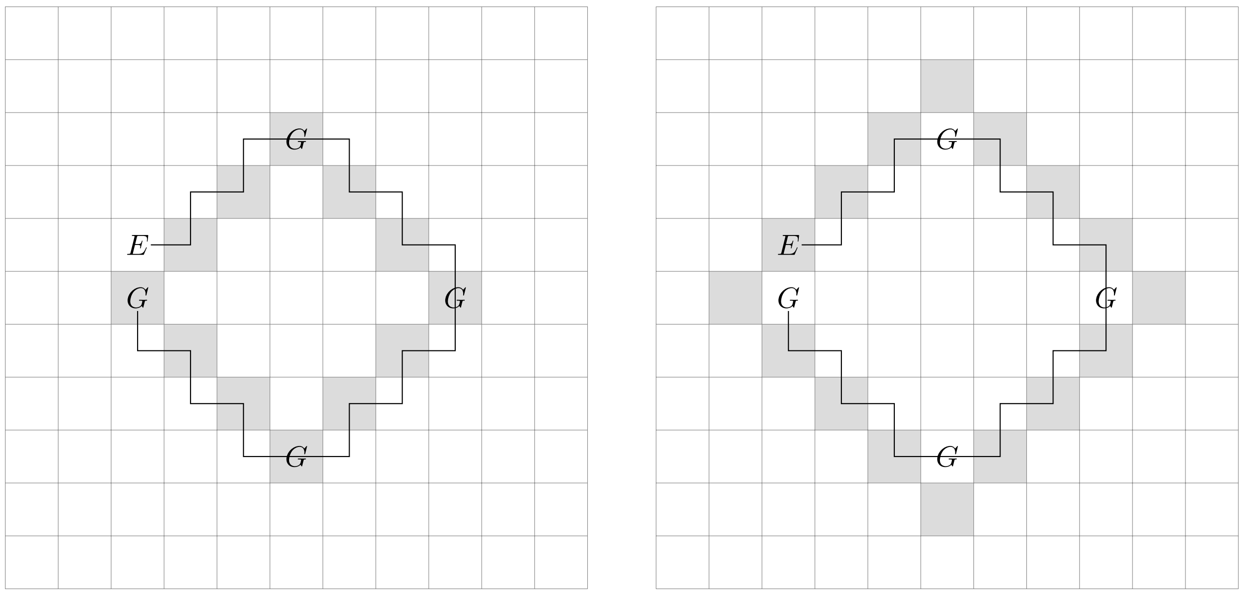


Figure 2.1. TEST

2.2. **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 1 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.

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.

2.3. **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 *Search Phase* begins with the first team exploring level 2. Their search begins with the explorer walking out along the west axis, directly followed by the west guide. The other guides walk in sync with the west guide, along their respective axes. Once the explorer reaches the point two steps to the left of the origin, 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, the second team begins to execute the *Search Phase.* 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 $\lfloor 2k/5 \rfloor$ levels being explored. However, if the treasure still has not been located, we need a way for the agents to continue searching.

2.4. Some Name.

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.

    Only one explorer searches each level, so only one explorer will touch any cell on the axis.

(2) The north guide from a new team is passing them.

    No team leaves the origin within twelve steps of another team, so there are at least twelve steps between any two north guides from new teams.

(3) The north guide from a team moving outwards to explore a different level is passing them.

    It takes 16 steps to explore level 2 and 32 steps to explore level 4. So, there are at least 16 steps between any two agents that have already searched a level.

From the above reasoning, if there are ever three consecutive time steps where the cell of a north guide waiting for an explorer is occupied by another agent, then it must be that the explorer passed through that cell.

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 the explorer for a team must leave the origin on an even time step, and the west guide from the team leaves the origin in the next time step (which is of course odd). 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, but this was not necessary for the north guide.

To remedy this, we first need the following claim.

Lemma 1. *Consider some west guide W waiting for its explorer to search a level. Before W's explorer returns, the explorer and west guide from some other team will pass through W's cell.*

Proof. Say that W's team is exploring level d. If $d = 2$, then W is on the first team and the claim is true by Assumption 3. Consider the case where $d > 2$. If W's team is exploring level d and no other search team has passed by W yet, then some team must be exploring level $d - 2$. But exploring level $d-2$ takes 16 steps less than exploring level d. So, the team exploring level $d-2$ will complete their exploration and move outwards before W's explorer returns. $\square$

Now, with this lemma in mind we modify the behavior of the west guide as follows. When the west guide reaches an empty cell, it stops there and waits as it had before. But now, once the west guide is passed by another team, it takes on step south, so that it sits just below the west axis.

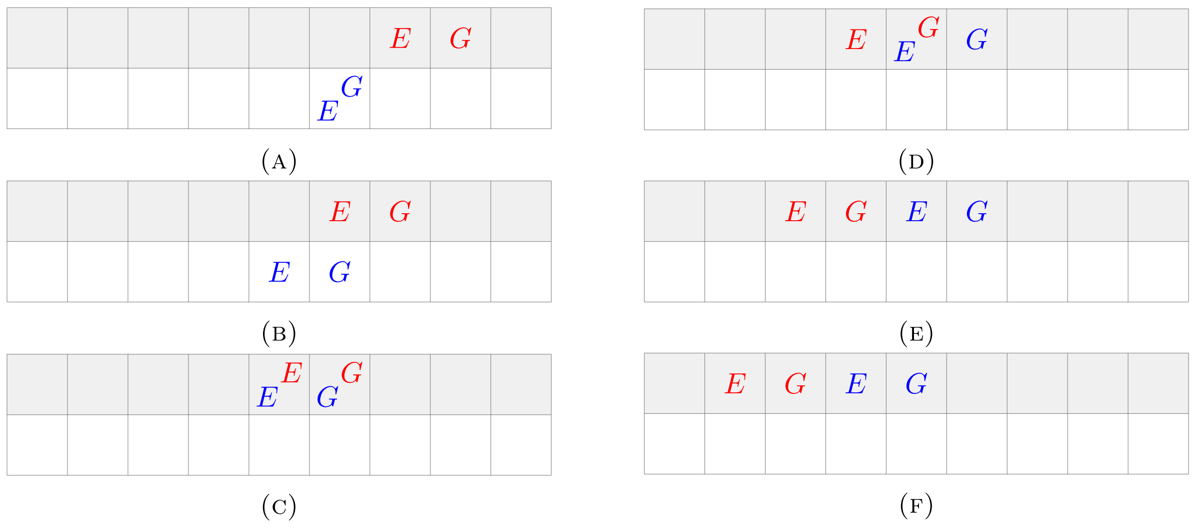


FIGURE 2.2

This will not 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 multiple teams walking in a row, in which case they may need to wait more than two 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.

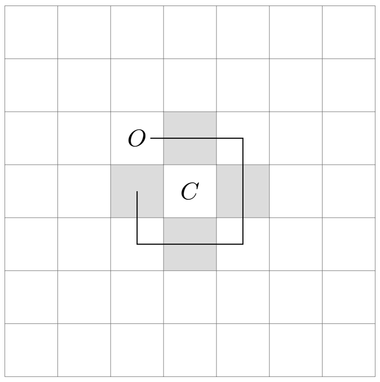


FIGURE 4.1

3. Separation Phase

In this section, we will describe the *Separation Phase* of the algorithm. This will be the first phase which is performed. At the start of this phase, all agents are located at the origin.

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.

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. Once an agent has returned to the origin, it has completed the *Separation Phase*, and moves directly into the ALLOCATION PHASE.

4. Allocation Phase

In this section we describe the ALLOCATION PHASE. There are two parts to this phase, *Part A* and *Part B*. Each agent completes either *Part A* or *Part B*. The agent decides which part to execute based on whether it is alone at the origin when it enters the *Allocation Phase*. If it is alone, it begins executing *Part B*. Otherwise, it begins executing *Part A*.

4.1. **Part B.**

The first few agents to enter the *Allocation Phase* will actually execute *Part A*. However, it is not clear why *Part A* is necessarily until we describe *Part B*, so we describe *Part B* first.

Say that an agent enters the *Allocation Phase* and finds that it is not alone at the origin. Then it begins to execute *Part B*. To do so, it follows 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.

Once the designated north agent has been passed by four agents, it knows that the rest of its team has passed, and takes a step north. At this point, that agent has completed the *Allocation Phase* and entered the *Search Phase*, so it behaves just like an agent in Section 2 having just left the origin.

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 and entering the *Search Phase*. 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 before moving south and entering the *Search Phase*.

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. Then, it waits for an agent to pass by it. Instead of entering the *Search Phase* at that time step, it waits one extra time step in the same place, and then moves left and enters the *Search Phase*.

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, also entering the *Search Phase*. 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 enter the *Allocation Phase* every other time step, it is impossible for another agent to encounter a west agent while it is waiting that extra time step before entering the *Search Phase*.

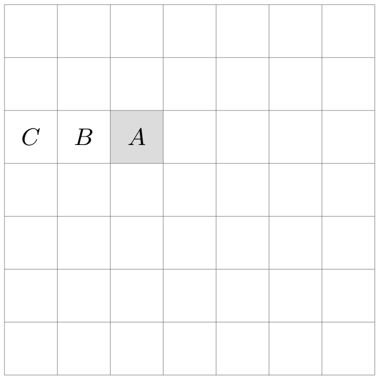
Figure 2.2 shows a sample execution of *Part B* on five agents. Since the *Separation Phase* involved randomness, we display an execution where we assume that the agents executed the *Separation Phase* in such a way that they end up entering the *Allocation Phase* exactly once every other time step. Note that the figure displays only even time steps.

At this point, we should note that we have shown that Assumptions 1 and 3 are valid *** MAKE EXTRA ASSUMPTION?? **. Each agent that enters the search team knows its role, and a team member of each role enters the *Search Phase* at least twelve units after the previous.

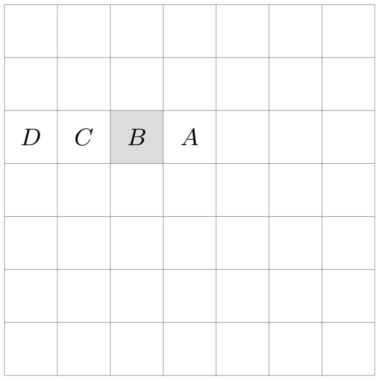
We are left with a final issue of dealing with the fact that currently, we need the agents to know whether they are the first team to enter the *Search Phase*. In Figure 2.2, we see that the guides all stop exactly two steps from the center, and the explorer begins searching level 2. However, they must 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.

4.2. **Part B.** This problem is solved by the introduction of *Part A*. Recall that an agent that has just entered the *Allocation Phase* will begin executing *Part A* if it finds itself alone at the origin. Since no agents from the *Separation Phase* can remain at the origin, the first agent to enter the *Allocation Phase* will immediately enter *Part A*. As we will see, an agent entering *Part A* immediately steps away from the origin. As a result, the next agent to enter the *Allocation Phase* will also be alone at the origin. This process will repeat until some *Part A* agent returns to the origin. At this point, all agents entering the *Allocation Phase* will begin executing *Part B*.

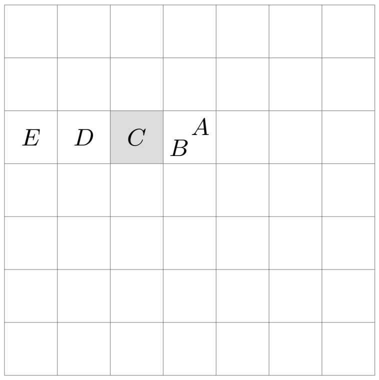
Part A is very similar to *Part B*. The agents will walk along a predetermined course. If they find themselves alone in certain cells, they will stop and wait there until some condition indicates to them that they can move forward. Figure xx shows that path that agents execute in *Part A*. As in Figure xx, the gray cells show the ones in which agents stop in if they find themselves there alone.



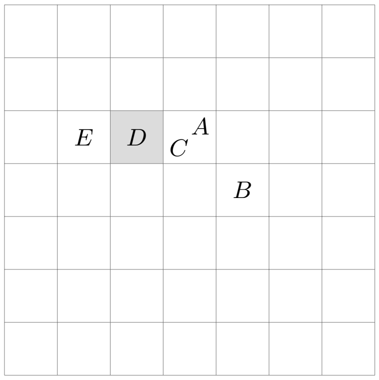
(A) $t = 0$



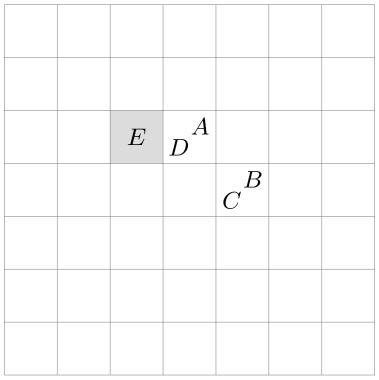
(B) $t = 2$



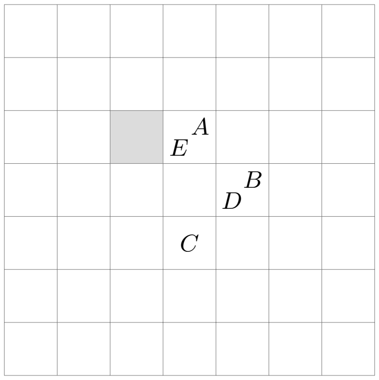
(C) $t = 4$



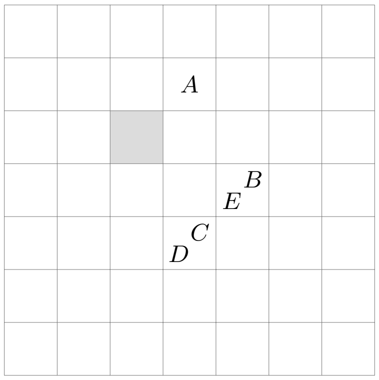
(D) $t = 6$



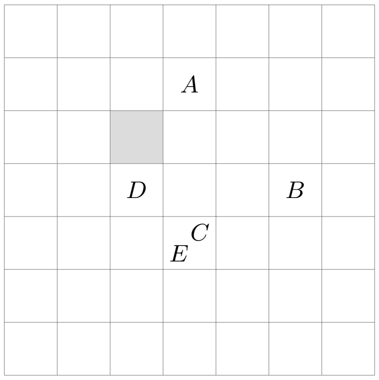
(E) $t = 8$



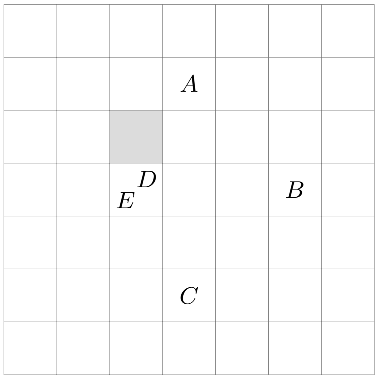
(F) $t = 10$



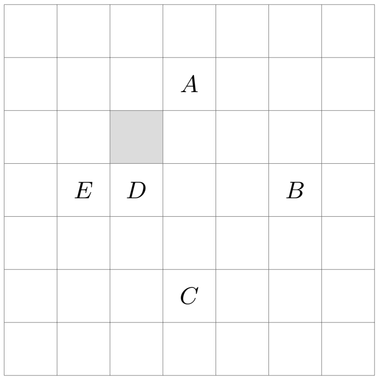
(G) $t = 12$



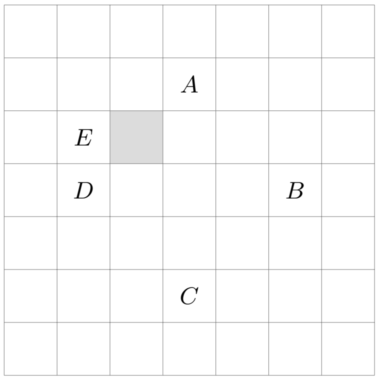
(H) $t = 14$



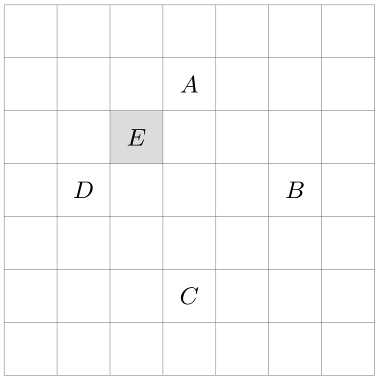
(I) $t = 16$



(J) $t = 18$



(K) $t = 20$



(L) $t = 22$

Figure 4.2. TEST

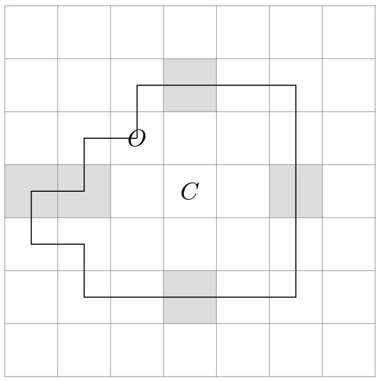


FIGURE 4.3

If an agent passes by all five gray cells and finds them occupied, then it continues along the path and returns to the origin. After this point, there will always be at least one agent at the origin, so all subsequent agents entering the *Allocation Phase* will execute *Part B*. We will also note that in the worst case (where agents are entering the *Allocation Phase* every other time step), there can be at most 13 agents which begin executing *part A* before one of them returns to the origin. This means at most 8 agents will return to the origin. Note that any agent who executes *Part A* and returns to the origin will never leave the origin again.

Now, when an agent enters *Part B*, it can just assume that it is not on the first team. This is because the *Part A* agents will still be positioned in their gray spots, and so even the first *Part B* 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 *Separation* and *Allocation Phases* at that time, we do not want them to re-search early levels after seeing the *Part A* agents positioned at level 2.

The real purpose of the *Part A* team is to ensure that the first *Part B* team does not wander away. So, once that team has passed, we can have the *A* agents enter the *Search Phase*. It remains to show how the *Part A* agents know the difference between the first search team passing by them, and just another *Part A* agent making its way around. To resolve this, we note that agents only enter *Part A* every other step.

When the first search team begins to move outwards, it will encounter agents at level 2. Only *Part A* agents will ever be stopped there, so it will know the agents are *Part A* agents. The new search team will then wait in the cells with the *Part A* agents for an extra time step before proceeding. This will signal to the *Part A* agents that they have been passed by a search team. After that search team steps away, the *Part A* agents can enter the *Search Phase*, with their roles determined by their positions.

Note that all level 1, 2, and 3 cells are searched by *Allocation Phase* agents except for the level 3 cells on the axes. Since these will be search by guides in the *Search Phase*, there is no need to ever have a team explore these levels. We have to be careful because no agent ever actually explores the center. To fix this, we just have all the agents move to the center and back to the origin before beginning the *Separation Phase.*

5. ANALYSIS

Theorem 2. 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 the *Separation Phase*. To do so, we use the following lemma.

Lemma 3. *After $O(t+\log k)$ time steps, at least t agents have completed the Separation Phase 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. □

Define team 0 to be the team which is formed in *Part A*, and define team i to be the i^{th} team formed in phase B. Next, we will define team j to be the last team to enter the *Search Phase*. Since the agents move in synchrony, the runtime of the algorithm is equal to the number of steps taken by team j.

Lemma 4. *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 *Part A* 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 5. *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 the *Separation* and *Search Phases* and participated in the *Search Phase*. By Lemma xx, it takes $O(k + \log k) = O(k)$ time steps with high probability before team m completes the *Separation Phase*. Note at after the final agent on a team completes *Separation Phase*, it only takes a constant number of steps before that team moves on to the *Search Phase*.

When a team enters the *Search Phase*, it moves out to find the first unexplored level. Say that this is level f. Once there, the team explores that level. Recall that it takes $O(d)$ steps to explore level d. So, in $O(f)$ steps, the team explores level f. Next, they move out and explore the next unexplored level, which will be level $f + 2m$. After exploring that level, they will move out and explore level $f + 4m$. This process can repeat at most $D/2m$ times. So, the total amount of work is given by the following sum.

$$f + \sum_{z=0}^{D/(2m)} O(f + z \cdot (2m)) + m$$

$$= f + O\left(\sum_{z=0}^{D/k} f + zk\right)$$

$$= f + O\left(\sum_{z=0}^{D/k} f + k\sum_{z=0}^{D/k} z\right)$$
$$= f + O((D/k) \cdot f + D^2/k)$$

To simplify this further, we claim that $f = O(D)$. To see why, we note that if $f > D$, 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 6. *If $j < m$, then the algorithm terminates after $O(D + \log k)$ steps with high probability.*

Proof. This means that not all teams completed the *Separation* and *Allocation Phases* by the time that some team located the treasure.

We know that team $j + 1$ never completes the *Allocation Phase*, because the algorithm terminates before this. By Lemma xx, this team would have completed the *Separation Phase* in $O(j + \log k)$ time steps with high probability. Since the *Allocation Phase* only involves a constant number of steps after the final agent of the team enters that phase, this means that team $j + 1$ also would have completed the *Allocation Phase* in $O(j + \log k)$. 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.

□