Exploration of High-Dimensional Grids by Finite Automata

Stefan Dobrev
Institute of Mathematics, Slovak Academy of Sciences, Bratislava, Slovakia
http://www.ifi.savba.sk/~stefan/
stefan@ifi.savba.sk

Lata Narayanan
Department of CSSE, Concordia University, Montreal, Canada
https://users.encs.concordia.ca/~lata/
lata@cs.concordia.ca

Jaroslav Opatrny
Department of CSSE, Concordia University, Montreal, Canada
https://users.encs.concordia.ca/~opatrny/
opatrny@cs.concordia.ca

Denis Pankratov
Department of CSSE, Concordia University, Montreal, Canada
https://users.encs.concordia.ca/~denisp/
denisp@cs.concordia.ca

Abstract

We consider the problem of finding a treasure at an unknown point of an n-dimensional infinite grid, $n \geq 3$, by initially collocated finite automaton agents (scouts/robots). Recently, the problem has been well characterized for 2 dimensions for deterministic as well as randomized agents, both in synchronous and semi-synchronous models [12, 21]. It has been conjectured that $n+1$ randomized agents are necessary to solve this problem in the n-dimensional grid [17]. In this paper we disprove the conjecture in a strong sense: we show that *three* randomized synchronous agents suffice to explore an n-dimensional grid for *any* n. Our algorithm is optimal in terms of the number of the agents. Our key insight is that a constant number of finite automaton agents can, by their positions and movements, implement a stack, which can store the path being explored. We also show how to implement our algorithm using: four randomized semi-synchronous agents; four deterministic synchronous agents; or five deterministic semi-synchronous agents.

We give a different algorithm that uses 4 deterministic semi-synchronous agents for the 3-dimensional grid. This is provably optimal, and surprisingly, matches the result for 2 dimensions. For $n \geq 4$, the time complexity of the solutions mentioned above is exponential in distance D of the treasure from the starting point of the agents. We show that in the deterministic case, one additional agent brings the time down to a polynomial. Finally, we focus on algorithms that never venture much beyond the distance D. We describe an algorithm that uses $O(\sqrt{n})$ semi-synchronous deterministic agents that never go beyond $2D$, as well as show that any algorithm using 3 synchronous deterministic agents in 3 dimensions, if it exists, must travel beyond $\Omega(D^{3/2})$ from the origin.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Multi-agent systems, finite state machines, high-dimensional grids, robot exploration, randomized agents, semi-synchronous and synchronous agents

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.139

Category Track C: Foundations of Networks and Multi-Agent Systems: Models, Algorithms and Information Management

Related Version A full version of the paper is available at https://arXiv.org/abs/1902.03693.

Funding Research supported by NSERC, Canada.

46th International Colloquium on Automata, Languages, and Programming (ICALP 2019).
Editors: Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi;
Article No. 139; pp. 139:1–139:16

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Motivated by the self-organizing behaviour of ants and other social insects, swarm robotics leverages the collective capability of a collection of extremely simple and inexpensive robots. Such robots have very limited computation and communication capabilities, and yet can collectively perform seemingly complex tasks such as: forage for food [14]; form patterns [26]; pull heavy objects [23]; and play *Für Elise* on the piano [15].

A series of recent papers [24, 22, 21, 12, 17] studies the conditions required for such primitive robots (also called agents or scouts) to search for a treasure placed at an unknown location in an infinite two-dimensional grid. In particular, they consider agents whose behaviour is controlled by a finite automaton (FA), who are equipped with a global compass, and can only communicate with other agents that are at the exact same grid location as themselves. Furthermore, this communication is limited to see the current state of other co-located agents. The primary question of interest is: *how many* such agents are needed to search for a treasure located at an unknown location in an infinite n-dimensional grid for $n \geq 2$? As shown in [12, 21] for $n = 2$, the answer depends on the computational power of the agents: whether or not they have access to random bits, the amount of memory they have, and whether or not they are synchronized. Note that for randomized algorithms, we require a finite mean hitting time for every node in the grid. The set of agents is *fully synchronous* if they operate by the same global clock; they are *semi-synchronous*[1] if in every time slot, a subset of adversarially scheduled agents is active. Full details of the agent models are given in Section 2.

The case of the 2-dimensional grid has been completely characterized. It has been shown that if the agents are deterministic and semi-synchronous, 4 agents are necessary [12] and sufficient [21]. If the agents are fully synchronous and deterministic, then 3 agents are necessary and sufficient [21]. In [17], the authors proved that 3 agents are necessary to search the 2-dimensional grid, even if they are fully synchronized and are randomized. They conjectured that in an n-dimensional grid, $n + 1$ agents would be necessary.

▶ **Conjecture 1** ([17]). *For $n \geq 3$, any search strategy on the n-dimensional infinite grid requires at least $n + 1$ agents.*

The main result of this paper is to disprove the above conjecture; we show that three randomized synchronous agents, or 5 deterministic semi-synchronous agents can explore any n-dimensional grid. These algorithms are completely different from previous algorithms for grid exploration, and are based on the key insight that a constant number of finite automata agents can, by their positions and movements, implement a stack that stores the path being explored.

1.1 Our results

Our main result is an algorithm for 3 randomized synchronous agents to explore an n-dimensional grid for any $n \geq 3$. This result is optimal, since 3 agents are necessary to explore even the 2-dimensional grid. Next we show how to "derandomize" the algorithm with the addition of one agent. If the agents are semi-synchronous, the algorithm can be implemented

[1] In some related literature [22, 17, 21] the same model was referred to as asynchronous. We follow the terminology of semi-synchronous of [12] and the vast literature on autonomous mobile robots to avoid confusion with a fully asynchronous model.

with the addition of one more agent, in both the randomized and deterministic cases. We also show that in the 3-dimensional grid, 4 deterministic semi-synchronous agents are sufficient for grid exploration.

The algorithms mentioned above have an exploration cost/time that is exponential in the volume of the smallest ball containing the treasure. In Section 5, we give an algorithm with exploration cost linear in the volume of the ball, which is similar to the algorithm for the 2-dimensional grid given in [21], but with an important modification that enables exploration of the 3-dimensional grid without increasing the number of agents. Our algorithm is optimal in the explored space and also in the number of agents, since 4 agents are necessary to explore even the 2-dimensional grid. In Section 4, we give a deterministic synchronous algorithm for exploring the n-dimensional grid that uses 5 agents and takes time polynomial in D, the distance from the origin to the treasure. A semi-synchronous implementation of this algorithm uses 6 agents. Table 1 shows our results.

Table 1 Exploration of an n dimensional infinite grid. Numbers marked with * indicate that the number of agents used or the exploration cost is optimal. We use c in the exploration cost to indicate a constant.

Model	Number of agents	Section	Exploration cost
Randomized Synchronous	3^*	Section 3.2	c^D
Randomized Semi-synchronous	4	Section 3.2	c^D
Deterministic Synchronous	4	Section 3.3	$O(2^{D+2n})$
	5	Section 4	$D^{O(n)}$
Deterministic Semi-synchronous	4^* $(n = 3)$	Section 5	$O(D^3)$ *
	5 $(n > 3)$	Section 3.3	$O(2^{3D+4n})$
	6 $(n > 3)$	Section 4	$D^{O(n)}$

In Section 6 we describe the following additional results. We give a lower bound of $\Omega(D^{3/2})$ on the distance from the origin that must be travelled by some agent in any 3-agent deterministic synchronous algorithm, and give an algorithm using $O(\sqrt{n})$ deterministic semi-synchronous agents in which no agent travels distance more than $2D$. Lastly, we extend our algorithms to agents without global compass. We show that one additional agent is sufficient in the semi-synchronous model, while two additional agents are sufficient in the synchronous model.

1.2 Related work

There is a lot of related literature on multi-agent systems and the exploration problem: from the early work on the cow-path problem to the more recent work on exploration of graphs, labyrinths, and grids by finite state agents [3, 6, 1, 2, 4, 5, 11, 16, 29, 32, 31, 34, 10, 19, 28, 8, 7, 13, 27, 9]. In this section we briefly mention the work that is most directly relevant to this paper.

The authors of [24] introduced the problem of k randomized mobile agents, starting from the same initial position, and searching for a treasure at an unknown location on the two-dimensional infinite grid. In their model, the agents are Turing machines, but cannot communicate at all. They show that if the agents have a constant approximation of k, the treasure can be found optimally in time $O(D + D^2/k)$, where D is the distance between the initial location and the treasure. The authors of [22] consider semi-synchronous and randomized FA agents and show that the same time complexity can be achieved. The relationship between the number of random bits available and the search time was studied in [33].

Emek et al. [21] posed the question of *how many* agents are required to find the treasure. They studied deterministic as well as randomized agents, synchronous as well as semi-synchronous agents, and FA agents, as well as agents that are controlled by a push-down automaton (PDA). They show that the problem can be solved by any of the following: 4 deterministic semi-synchronous FA agents; 3 deterministic synchronous agents; 3 randomized semi-synchronous FA agents; 1 deterministic FA together with 1 deterministic PDA agent; 1 randomized PDA agent. On the negative side they show that the problem cannot be solved by 2 deterministic (synchronous) FA agents; a single randomized FA agent; a deterministic PDA agent. Cohen et al. [17] prove that at least 2 FA agents are necessary to explore the one-dimensional grid and at least 3 FA agents are needed to explore the two-dimensional grid, thus proving the optimality of the FA-agent deterministic synchronous and randomized semi-synchronous algorithms in [21]. Recently it was shown that 3 deterministic semi-synchronous FA agents cannot perform exploration of the 2-dimensional grid [12], thus proving the optimality of the 4 FA-agent deterministic synchronous algorithm in [21].

A large body of work is devoted to the capabilities of autonomous mobile robots with very limited computational and communication abilities; see [25] for a comprehensive introduction. While we use some of that terminology in this paper, their robots are usually assumed to be identical, anonymous, and communication is limited to being able to "see" each other's positions, regardless of how far they are. In contrast, in our model, the robots follow different algorithms (this can be done by having different initial states of the same FSM), and only see other robots if they are at the same location, and they can exchange a message with the collocated robots. Equivalently, they can be assumed to see the current states of other robots at the same location. This is similar to the "robots with lights" model in the autonomous mobile robot literature [18].

2 Model and Notation

We use the same models (with the exception of Theorem 10 on agents without a global compass) as in [21, 12]. For completeness, we recall key definitions and introduce some notation in this section.

Our search domain is $\mathbb{Z}^n$ with the *Manhattan metric*, i.e., the distance between two points $p, p' \in \mathbb{Z}^n$ is defined as $||p-p'|| = \sum_{i=1}^{n} |p_i - p'_i|$. We refer to $\mathbb{Z}^n$ as the *n-dimensional integer grid* and its elements as *grid points, points, or cells.* A grid point $p = (p_1, p_2, \ldots p_i, \ldots, p_n)$ is *adjacent* to every grid point $(p_1, p_2, \ldots p'_i, \ldots, p_n)$, where $|p_i - p'_i| = 1$ for some i, $1 \leq i \leq n$. We assume that any two grid points cannot be distinguished from each other by an agent, and that includes the origin from which the search starts.

The search for the treasure in the grid is done using a fixed number of agents, each modelled by a finite automaton. These finite automata can be the same, except they typically have a different initial state. Two agents can exchange information with each other only when they occupy the same grid location at the same time. We assume that all agents have the same global n-dimensional compass. Initially, all agents are located in the same grid point, assumed without loss of generality to be the origin of the grid. The treasure is located at distance D from from the origin, where D is unknown to the agents.

Time is divided into discrete units. In each time unit an *active* agent performs a single *look-compute-move* cycle. In the look part of the cycle the agent sees the state of other agents located in its own grid point. In the compute part of the cycle the agent determines,

using its own state and those it sees, to which adjacent node to move to, if at all. The agent also determines its new state. Such a move is then executed in the move part of the cycle. When we consider randomized algorithms, we assume that an agent has access to an infinite one-way tape with i.i.d. random bits. We say that the system is *synchronous* if at each time unit all agents are active. We say that the system is *semi-synchronous* if at each time unit only a subset of agents, chosen by an adversarial scheduler, is active. The adversarial scheduler must schedule each agent infinitely often.

In addition to the question of whether $\mathbb{Z}^n$ can be fully explored by k agents, we are also interested in the efficiency of such exploration procedures. We refer to this measure of interest as *the exploration cost*. Intuitively, we measure how long it takes for k agents to visit all $\Theta(D^n)$ points in a sphere of radius D. In the synchronous model, this measure is simply the number of time units taken by the agents to complete the exploration. In the semi-synchronous case, because of adversarial scheduling, the exploration cost is defined as the total distance travelled by all robots to visit all points in a sphere of radius D. Since the number of robots is constant, we could as well define the exploration cost as the maximum path length travelled. Now that we have discussed this subtlety, we will abuse the terminology and use "exploration cost" and "time" interchangeably.

3 Exploration of n-dimensional Grids

A straightforward generalization of the algorithms for the exploration of 2D grids in [21] to n dimensions results in algorithms that use $\Omega(n)$ agents. Consider, for example, such a simple generalization of a randomized 2D algorithm. The basic idea of the $n+1$-agent randomized algorithm for n dimensions is to make an n-segment walk, starting from the origin, and walking the i-th segment along dimension i. The lengths of the segments are chosen randomly, and one agent per segment is used to mark its endpoint. This allows the agent to find the way back to the origin and start another random trial. In essence, this algorithm uses 2 agents per dimension to store in unary the distance travelled in this dimension, and by an appropriate arrangement we can reuse one of the agents in the successive dimension to bring the number of additional agents per dimension to 1.

The main idea of our approach is a realization that it is not necessary to use $n+1$ agents to store n numbers of segment lengths. Observe that segment lengths are stored and retrieved in this randomized algorithm in the first-in last-out order. Thus this algorithm can be realized if we can store the agent's movements in a stack. It turns out that we can use a *constant* number of agents, independent of the grid's dimension, to *implement a stack* in which the active agent, that does the exploration, stores its walk and subsequently uses to return to the origin. The active agent "carries" the stack along its walk, i.e., it always makes the agents representing the stack to shift by one position in the direction it moves before making that move itself in its walk.

3.1 The Stack Implementation

The format of data stored in the logical stack is the string $\alpha \in (0^*1)^n$, where 0 represents *continue walking in the current direction*, 1 represents *switch to the next dimension*.
The physical implementation of the stack stores this data by interpreting α^r (that is α reversed) as a binary number S and storing it in unary as a distance between two agents located in a row in the first dimension.

We employ the following agents:

- a: the *active* agent that is doing the exploration of the grid; in the semi-synchronous model this is the only agent moving around and manipulating the other agents,
- b: the *base* of the stack, from which measurements are taken, and representing the current logical location of the exploration,
- c: the *counter* agent; this is an auxiliary agent for implementing the stack operations in the semi-synchronous model,
- d: the *distance* agent; its distance from the base b stores the content of the stack in unary,
- e: the *extra* agent used in the deterministic algorithms to store an extra copy of the current stack value.

The basic stack operations we need to implement are *isEmpty()*, *push(v)* where $v \in \{0, 1\}$ and *pop()*. Operation *isEmpty()* simply returns whether b and d are collocated. Implementation of *push()* and *pop()* is model-dependent and given below.

3.1.1 Implementing Semi-Synchronous Stack

Algorithms 1 and 2 show the implementation of push and pop operations for the semi-synchronous stack. Notice that after each push/pop operation the agents b and c in these algorithms are not only collocated, but they actually return to the position they had before push/pop.

Algorithm 1 Semi-synchronous stack: push(v).

On entry: b and c collocated, a and d collocated at $b + Se_1$.
On exit: b and c collocated, a and d collocated at $b + (2S + v)e_1$.
procedure PUSH(v)
    a goes to b and brings c to d
    while b and d are not collocated **do**
        a goes to c, pushes it one step away from b and returns to d
        a pushes d one step closer to b
    end while
    d becomes c
    a goes to c and tells it to become d
    if v=1 **then**
        a pushes d one step away from b
    end if
end procedure

Algorithm 2 Semi-synchronous stack: pop().

On entry: b and c collocated, a and d collocated at $b + Se_1$.
On exit: b and c collocated, a and d collocated at $b + \lfloor S/2 \rfloor e_1$, returns $S \bmod 2 = 1$.
procedure POP
    while b and c are at distance more than 1 **do**
        a pushes d one step closer to b
        a goes to c and pushes it one step away from b
    end while
    $v = d$ is one step from c
    c and d switch roles
    a brings c to a and returns to d
    return v
end procedure

3.1.2 Implementing Synchronous Stack

In the synchronous model, we can synchronize the movements of agents to effectively multiply or divide the stack content by 2 without the need of the counter agent c, see Figure 1.

Algorithm 3 Synchronous stack: push(v).

1: On entry: a and d collocated at $b + Se_1$.
2: On exit: a and d collocated at $b + (2S + v)e_1$.
3: **procedure** PUSH(v)
4: a goes to b and then back towards d until they meet, walking at speed 1
5: d walks away from b at speed 1/3 (move, wait, wait, see Figure 1)
6: **if** v=1 **then**
7: a pushes d one step away from b
8: **end if**
9: **end procedure**

Algorithm 4 Synchronous stack: pop().

1: On entry: a and d collocated at $b + Se_1$.
2: On exit: a and d collocated at $b+\lfloor S/2 \rfloor e_1$, returns $S \bmod 2 = 1$.
3: **procedure** POP
4: a goes to b and then back towards d until they meet, walking at speed 1
5: d walks towards b at speed 1/3 (move, wait, wait, see Figure 1)
6: **if** a and d meet right after d's move **then**
7: return 1
8: **else**
9: return 0
10: **end if**
11: **end procedure**

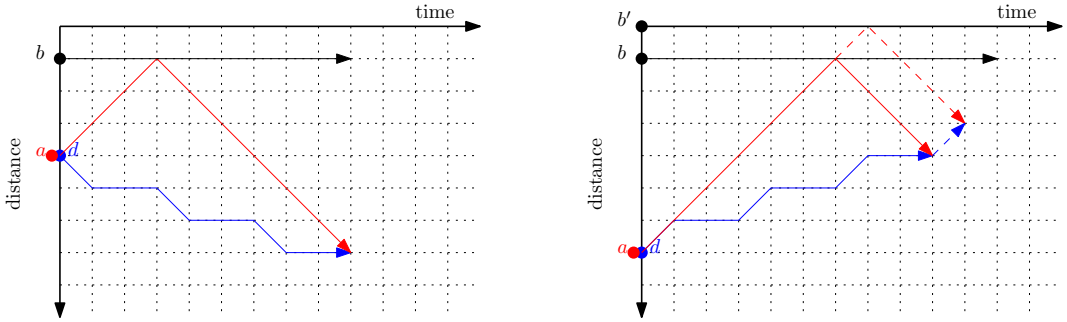

Figure 1 Multiply/divide operations by synchronous agents, shown in time-space diagrams. At left: multiplication by 2 by is done by agent a walking up to b and then down, while agent d walks down at speed 1/3. When a and d meet they have doubled the distance to b. At right: division by 2 is shown, in this case a and d walk towards b at speed 1 and 1/3 respectively; for even case a turns at b else at one node above b, and then walks back until meeting d.

3.2 The Randomized Algorithm

As already stated in the initial part of this section, the main idea of the algorithm is to use the stack to store the random choices during the walk, so that the agent can return to the origin. The agent a "carries" the stack along this walk so that the operations can be applied without the need to search for the stack.

In addition to the stack methods, it uses two new procedures. Procedure *random(p)* returns 1 with probability p, while *moveStack()* moves the whole stack one step in the direction specified. Note that since the whole stack is located on a single line, this can be accomplished by agent a walking to each of the other agents and instructing them to move one step in the specified direction.

The algorithm works in rounds, that we number $1, 2, 3, \ldots$, that correspond to the iteration numbers of the outer while loop. At the beginning of each round, the active robot picks a binary string $R \in \{-1, 1\}^n$ uniformly at random. This string indicates that the robot is

going to explore dimension i in direction R_i. Then for each dimension i from 1 to n, the active robot travels for $Z_i - 1$ steps in direction R_i, where Z_i is geometrically distributed with parameter p (to be determined later). Note that we want Z_i to represent the length of the string pushed onto the stack while moving in dimension i. Since the string pushed on the stack includes the "separator" between dimensions, we have the -1 term for the actual number of moves. We call the concatenation of all such moves over all dimensions *the logical path of the active robot*. If no treasure is found, the active robot uses the stack to retrace its logical path back to the origin by travelling $Z_n - 1$ steps in direction $-R_n e_n$ first, followed by $Z_{n-1} - 1$ steps in direction $-R_{n-1} e_{n-1}$, and so on. To compute the exploration cost of each round, we need a simple helper lemma.

Algorithm 5 Randomized Grid Exploration.

```
1:  while treasure not found do
2:      Pick a random n-bit string R ∈ {−1, 1}^n
3:      for i = 1 to n do
4:          while random(p) = 0 do
5:              push(0)
6:              moveStack(R_i e_i)
7:          end while
8:          if i < n then
9:              push(1)
10:         end if
11:     end for
12:     i = n
13:     while not empty() do
14:         while pop()= 0 do
15:             moveStack(−R_i e_i)
16:         end while
17:         i = i − 1
18:     end while
19: end while
```

▶ **Lemma 2.** *Let S be the maximal stack size during one iteration of the outer while loop of Algorithm 5. The overall cost of this iteration is $O(S^2)$ when implemented by semi-synchronous agents, and is $O(S)$ when implemented by synchronous agents.*

Proof. In the semi-synchronous model, each push() or pop() costs $O(X^2)$, where X is the actual stack size, as the active agent zig-zags between b and d. On the other hand, in the synchronous model, the cost of each operation is linear in the stack size. The cost of moving the stack is linear in both models.

As the stack size grows exponentially, and then reduces exponentially, the overall cost is determined by the cost when the stack is the largest, i.e. $O(S^2)$ and $O(S)$ for the semi-synchronous and synchronous models, respectively. ◀

Observe that during a given round the maximum size of the stack is $2^{Z_1+\cdots+Z_n}$. Thus the exploration cost of each round is at most

$$2(Z_1 + \cdots + Z_n)2^{\Theta(Z_1+\cdots+Z_n)}$$

where $2(Z_1 + \cdots + Z_n)$ is the bound on the overall length of the logical path (there and back) of the active robot, and by Lemma 2 each step of the active path costs $2^{\Theta(Z_1+\cdots+Z_n)}$, since we need to perform operations on the stack of size $2^{Z_1+\cdots+Z_n}$. Also note that

$$2(Z_1 + \cdots + Z_n)2^{\Theta(Z_1+\cdots+Z_n)} = 2^{\Theta(Z_1+\cdots+Z_n)}.$$

Let c be the constant in the Θ notation such that the exploration cost of a round is at most $2^{c(Z_1+\cdots+Z_n)}$.

For simplicity, we will assume that the active robot checks for the treasure only at the far end-point of the logical path in each round. This assumption might lead to a more pessimistic upper bound on the exploration cost than if we assumed that the active robot checks for treasure at each grid point that it visits. However, our assumption simplifies the calculations and is sufficient for our purposes.

▶ **Theorem 3.** *Algorithm 5 locates the treasure in the n-dimensional grid in finite expected time, using either 4 semi-synchronous or 3 synchronous agents.*

Proof. Consider the infinite sequence of random variables $(X_i)_{i=1}^{\infty}$, where X_i is the exploration cost of round i. Note that the X_i are independent and identically distributed. Consider the exploration cost of a particular round, e.g., X_1. Then we have $X_1 \leq 2^{c(Z_1+\cdots+Z_n)}$, where the Z_i and c are as defined above. Therefore:

$$\begin{aligned}
\mathbb{E}(X_1) &\leq \mathbb{E}\left(2^{c(Z_1+\cdots+Z_n)}\right) \\
&= \sum_{i_1=1}^{\infty}\sum_{i_2=1}^{\infty}\cdots\sum_{i_n=1}^{\infty} 2^{c(i_1+\cdots+i_n)} p^{i_1-1}(1-p)p^{i_2-1}(1-p)\cdots p^{i_n-1}(1-p) \\
&= \left(\sum_{i_1=1}^{\infty}(2^c p)^{i_1-1}2^c(1-p)\right)\left(\sum_{i_2=1}^{\infty}(2^c p)^{i_2-1}2^c(1-p)\right)\cdots\left(\sum_{i_n=1}^{\infty}(2^c p)^{i_n-1}2^c(1-p)\right) \\
&= 2^{cn}(1-p)^n\frac{1}{(1-2^c p)^n},
\end{aligned}$$

where the last step holds as long as $2^c p < 1$ that is $p < 1/2^c$.

Define a random variable T to be the minimum t such that the far end-point of X_t coincides with the treasure. That is, our exploration procedure terminates in round T, but not earlier. Suppose that the treasure is located at position $(k_1, \ldots, k_n)$ where $|k_1| + \cdots + |k_n| = D$. By the discussion immediately preceding the statement of this theorem, the probability that the treasure is found in a particular round is $\widehat{p} = 2^{-n}(1-p)^n p^{k_1}\cdots p^{k_n} = 2^{-n}(1-p)^n p^D$, where 2^{-n} is the probability of guessing correctly the signs of the k_i and $p^{k_i}(1-p)$ is the probability of travelling the correct number of steps in dimension i. Thus T is geometrically distributed with parameter $\widehat{p}$. Therefore, $\mathbb{E}(T) = 1/\widehat{p}$.

We are interested in bounding the overall exploration cost, that is, $\mathbb{E}(X_1 + \cdots + X_T)$. Since the X_i are i.i.d. and T is a stopping time, it follows by a generalization of Wald's equation [35] to stopping times that $\mathbb{E}(X_1 + \cdots + X_T) = \mathbb{E}(T)\mathbb{E}(X_1) \leq \frac{1}{\widehat{p}}2^{cn}(1-p)^n\frac{1}{(1-2^c p)^n} < \infty$. This holds as long as we choose $p < 1/2^c$. Since c is a constant, such a probabilistic coin can be implemented by a finite automaton. The statement of the theorem follows by the number of robots sufficient to implement stack operations in each of the models (synchronous vs. semi-synchronous). ◀

3.3 The Deterministic Algorithm

The main idea is to exhaustively go over all possible stack contents in increasing order, interpreting each stack as a specification of a walk. We also keep a backup of the initial stack content, and at the end of the walk we use the backup to return to the origin. The back-up stack is stored using an additional agent. The backup is needed, as reading the stack content during the walk destroys it. Note that after the outward walk, we do not logically reverse the stack; hence the return to the origin does not use the same path as the original walk. However, this is not a problem as the walks along different dimensions are commutative.

Finally, we should mention that some generated stacks do not necessarily have the correct format, some may contain too few or too many 1s. However, this is easy to handle by the algorithm: too few ones just means we walked without using all of the dimensions, which is still a perfectly valid walk. The excessive 1s are simply ignored by taking the first excessive 1 as a directive to end the walk and return to the origin.

Using essentially the same arguments as in Lemma 2 yields:

▶ **Lemma 4.** *The cost of procedure Walk is $O(S^2)$ and $O(S)$ in the semi-synchronous and synchronous models, respectively, where S is the size of the backup stack.*

▶ **Theorem 5.** *Algorithm 6 locates the treasure in the n-dimensional grid with: (a) 5 agents and the exploration cost of $O(2^{3D+4n})$ moves in the semi-synchronous model, and (b) 4 agents and the exploration cost of $O(2^{D+2n})$ in the synchronous model.*

Algorithm 6 Deterministic Grid Exploration.

```
 1: while treasure not found do
 2:     Increment the backup stack
 3:     for every n-bit string R ∈ {−1, 1}^n do
 4:         execute Walk(R, 1)
 5:         execute Walk(R, −1)
 6:     end for
 7: end while
 8:
 9: procedure Walk(R, s)
10:     Restore stack from backup
11:     i = 1
12:     while not empty() and i ≤ n do
13:         while pop()= 0 do
14:             moveStack(sR_i e_i)
15:         end while
16:         i = i + 1
17:     end while
18: end procedure
```

Proof. The number of agents and the correctness follows easily from the construction.

It remains to sum up the cost of all calls to procedure Walk. Note that each point in space uniquely specifies a valid (i.e. with precisely n 1's) stack. Hence, the valid stack for the treasure at distance D contains $D+n$ digits. Therefore, the overall cost of Algorithm 6 is $2^n \sum_{X=1}^{2^{D+n}} O(X^2) = O(2^n(2^{D+n})^3) = 2^{3D+4n}$ in the semi-synchronous model, and $O(2^{D+2n})$ in the synchronous model (the initial 2^n covers all choices for string R). ◀

4 Polynomial time solutions

While designing our exploration algorithms in the previous section, we concentrated on minimizing the number of agents used, and the resulting cost of these algorithms is exponential in the volume $V(D)$, the smallest ball containing the treasure. A natural question to ask is whether this is an unavoidable consequence of using only a constant number of agents in the exploration. In this section we show that this is not the case: a single additional agent is sufficient to bring the cost of exploration down to a polynomial in $V(D)$.

The main reason the cost of algorithms in the preceding section is exponential is the number of incorrect stack contents being considered: as D grows compared to the fixed n, ever larger proportion of stack contents does not have the correct format and they result in repeatedly reaching already explored vertices. To avoid this problem we will efficiently explore an n-dimensional cube q^n of side q centered at the origin. We use again the stack idea to trace the exploration of q^n. The logical stack content now consists of n numbers in q-ary alphabet, describing a location within this cube. However, in this case, we also need to store the scale q. As before, the stack implementation interprets the logical content as a q-ary number and stores it in unary[2]. Since q also needs to be stored on its own, this incurs the additional cost of one agent. However, this allows us to multiply and divide by q, which would not have been possible without the extra agent.

The stack is manipulated using the explicit commands: isDivisible() which checks the divisibility by q; push(0) which multiplies the stack content by q; pop() which divides the stack content by q; and increment() which increments the top of the stack.

4.1 Stack operations: semi-synchronous implementation

In addition to agents a, b and d, we use agent f to maintain the value of q by placing it at $b + qe_1$. Furthermore, two counter agents c_d and c_f are used. At the beginning of the stack operations, f and c_f are collocated, as are b and c_d, and a and d. The basic procedure is a traversal of the whole stack by agent a, manipulating the tokens according to the specific command.

In push(0) (i.e. multiplying the stack content by q), a pushes c_f towards b and c_d away from b. Whenever c_f reaches b, a transports it back to f as well as pushes d one step closer to b. The process terminates when d reaches b; subsequently c_d and d change roles. The detailed procedure is given in Algorithm 8 in [20]. .

In isDivisible(), a pushes c_f towards b and c_d towards d, until c_d reaches d. Whenever c_f arrives to b, it is transported back to f. isDivisible() returns true iff at the moment when c_d reaches d, c_f is at b (or f).

Operation pop() means dividing the stack by q. The process is essentially reverse of push() – in every iteration/traversal of the stack, c_f and d are pushed towards b. Whenever c_f reaches b, it is brought back to f and c_d is pushed away from b. When d reaches b, c_d and d exchange their roles.

The detailed pseudocode of isDivisible() and pop() are straightforward and omitted. It is easy to see that the total cost of each of the stack operations is bounded by $O(S^2)$ where S is the maximal size of the stack during the operation.

[2] This is similar to the simulation of PDAs by counter machines – see Chapter 8.5 in Hopcroft, Motwani, and Ullman text [30]; however, the details of our implementation are completely different.

4.2 Stack operations: Synchronous implementation

A straightforward application of the technique from Section 3 would need agents travelling at speed $\frac{1}{2q+1}$ (for multiply) and $\frac{q-1}{s+q}$ (for divide), which is impossible with finite state agents.

Instead, we take q to be a power of two and implement the operation of multiply, divide by q via repeated applications of multiplication by 2, division by 2, respectively. Thus in this case f is placed at distance $\log q$ from b, instead of placing it at distance of q from b. The counter c_f is used to count the number of multiplications/divisions already performed, while the counter c_d is not used at all, i.e. only agents a, b, d, f and c_f are needed. The operations of doubling and halving were already described in Section 3 and shown to take $O(S)$ time. Since these operations are performed $\log q$ times, the total time complexity of every stack operation is $O(S \log q)$.

4.3 Fast deterministic grid exploration

Our polytime deterministic grid exploration algorithm is described in Algorithm 7. Starting with $q = 2$, and for any fixed value of q, the algorithm generates and visits the addresses (n-tuples from a q-ary alphabet) in lexicographic order. Then the agent a moves to position $(-q, -q, \ldots, -q)$, doubles the value of q, and moves on to the next iteration. Agent a always drags the stack along as it performs the exploration. The procedure $explore(i)$ is a recursive procedure to generate n-tuples in lexicographic order; it is called with logical stack content an i-tuple $x0$. It then iteratively calls $explore(i+1)$ to visit the $(n-i)$-dimensional cube of side q with $(x, j, 0, \ldots, 0)$ as the origin, for j ranging from 0 to $q-1$.

Note that the algorithm as shown in Algorithm 7 is presented using recursive calls for convenience; however, i is maintained in the local state.

▶ **Theorem 6.** *Let $V(D)$ be the volume of the ball of diameter D in the n-dimensional grid. Algorithm 7 locates the treasure in the n-dimensional grid with: (a) 6 agents and the exploration cost of $O(V(D)^3)$ moves in the semi-synchronous model, and (b) 5 agents and the exploration cost of $O(V(D)^2 \log D)$ in the synchronous model.*

Proof. The number of agents and the correctness follows easily from the construction. It remains to sum up the cost of all stack operations on a stack of size S. As already described, the cost of each stack operations is $O(S^2)$ and $O(S \log q)$ in the semi-synchronous and synchronous models, respectively. The maximal stack size S is bound by q^n, which is also the number of points covered by the stack base during one iteration of the outer loop (i.e. for fixed q). This results in the overall cost of $O(S^3)$ and $O(S^2 \log q)$ in the semi-synchronous and synchronous models, respectively. As q grows exponentially, the overall cost is determined by the cost for the last value of q. Finally, it is known that $V(D) = \frac{2^n}{n!} D^n$. As $q < 4D$ (the treasure would had been found if $q \geq 2D$), we get that $S \leq (4D)^n = 2^n n! V(D)$, where n is a constant. This proves the theorem. ◀

5 Exploring 3-dimensional Grids using 4 Semi-Synchronous agents

Our algorithm for the $3D$ grids explores the *sphere* consisting of all points at distance q from the origin for $q = 1, 2, 3, \ldots$, until reaching the sphere containing the treasure. In the Manhattan metric, the points of such a sphere are located on 8 triangular faces of a regular octahedron whose edges contain $q+1$ grid vertices.

The key to our success is an algorithm for exploring one such triangle using four agents, in such a way that (i) the value of q is maintained by the distance between some of the agents while exploring a triangle, so that it can be used for the exploration of all triangles of the

octahedron, (ii) the exploration of all eight triangles can be done in a fixed order, and (iii) the value of q can be increased for the exploration of the larger sphere after the exploration of the sphere of radius q is finished.

The detailed description of this algorithm and the proof of the following theorem appear in the full arXiv version of this paper [20].

▶ **Theorem 7.** *Assume that the treasure is located in a 3D grid at distance D from the origin. Algorithm* Explore3Dgrid *finds the treasure using 4 semi-synchronous agents, with the exploration cost of $O(D^3)$. This is optimal as far as the number of semi-synchronous agents used, and up to a constant factor in the exploration cost.*

6 Additional Results

In this section, we list without proofs three additional results mentioned in Section 1. We point out the sections of the arXiv version of this paper [20] where the details are given.

▶ **Theorem 8.** *Finding a treasure at distance D in an n-dimensional grid can be achieved with $O(\sqrt{n})$ agents, exploration cost $O(D^{n+\sqrt{n}})$, and without agents venturing further than distance $2D$ away from the origin (Section 6.1 of [20]).*

▶ **Theorem 9.** *Consider 3 deterministic synchronous agents that run a protocol for exploring $\mathbb{Z}^3$. In order to visit all grid points in the ball of radius D the distance of some agent from the origin must have been $\Omega\left(D^{3/2}\right)$ (Section 6.2 of [20]) .*

▶ **Theorem 10.** *Every algorithm in this paper which assumes agents with a global compass can be extended to work with agents without it by using one additional agent in the semi-synchronous model, and two additional agents in the synchronous model (Section 7 of [20]).*

7 Conclusions and Open Questions

We studied the exploration of n-dimensional grids for $n \geq 3$ by finite state automata agents. We showed the surprising result that three randomized synchronous agents suffice to find a treasure in an n-dimensional grid for any n; this is optimal in the number of agents. Our strategy can also be implemented by four randomized semi-synchronous agents, or four deterministic synchronous agents, or five deterministic semi-synchronous agents. For the three-dimensional case, we gave a different algorithm for the deterministic semi-synchronous case that uses only 4 agents, and is optimal. Our algorithms for $n \geq 4$ require agents to travel far away from the origin, i.e., exponential in D distance away, while looking for a treasure which is located at distance D from the origin. We also considered the question of whether it is possible to design algorithms that use few agents and do not require travelling much further than distance D away from the origin in order to explore the entire ball of radius D around the origin. We answered the question positively by describing an algorithm that uses $O(\sqrt{n})$ semi-synchronous deterministic agents that never travel beyond $2D$ while exploring the ball of radius D. We also showed that 3 synchronous deterministic agents in 3 dimensions performing search, if such an algorithm exists, must travel $\Omega(D^{3/2})$ away from the origin.

Algorithm 7 Fast Deterministic Grid Exploration.

```
1: q = 2
2: push(0)
3: while treasure not found do
4:     explore(1)
5:     moveStack(−q ∑_{i=1}^{n} e_i)
6:     q = 2q
7: end while
8:
9: procedure EXPLORE(i)
10:    if i > n then
11:        return
12:    end if
13:    repeat
14:        push(0)
15:        explore(i + 1)
16:        increment()
17:        moveStack(e_i)
18:    until isDivisible()
19:    pop()
20:    moveStack(−q e_i)
21: end procedure
```

Many interesting questions about the exploration of the n-dimensional grids remain open. Is it possible for 4 deterministic semi-synchronous agents to explore an n-dimensional grid for $n \geq 4$? For $n \geq 3$, can exploration of an n-dimensional grid be achieved by 3 randomized semi-synchronous agents or deterministic synchronous agents? What is the minimum number of agents that achieve polynomial time exploration? What is the minimum number of agents such that the distance of the furthest visited node from the origin is limited to polynomial in D? Is it possible to save an agent in the case of synchronous unoriented grids?

References

1 R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins. Searching with Uncertainty (Extended Abstract). In *Proceedings of the 1st Scandinavian Workshop on Algorithm Theory (SWAT 88)*, pages 176–189, 1988.

2 R. A. Baeza-Yates and R. Schott. Parallel Searching in the Plane. *Computational Geometry*, 5:143–154, 1995.

3 A. Beck. On the linear search problem. *Israel Journal of Mathematics*, 2(4):221–228, 1964.

4 A. Beck. More on the linear search problem. *Israel J. of Mathematics*, 3(2):61–70, 1965.

5 A. Beck and P. Warren. The return of the linear search problem. *Israel J. of Mathematics*, 14(2):169–183, 1973.

6 R. Bellman. An optimal search. *SIAM Review*, 5(3):274–274, 1963.

7 M. A. Bender and D. K. Slonim. The power of team exploration: two robots can learn unlabeled directed graphs. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 75–85, November 1994. doi:10.1109/SFCS.1994.365703.

8 Michael A. Bender, Antonio Fernández, Dana Ron, Amit Sahai, and Salil Vadhan. The Power of a Pebble: Exploring and Mapping Directed Graphs. *Information and Computation*, 176(1):1–21, 2002. doi:10.1006/inco.2001.3081.

9 M. Blum and W.J. Sakoda. On the capability of finite automata in 2 and 3 dimensional space. In *Proceedings of FOCS*, pages 147–161, 1977.

10 P. Bose, J.-L. De Carufel, and S. Durocher. Revisiting the Problem of Searching on a Line. In *ESA 2013, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, pages 205–216, 2013.

11 S. Brandt, K.-T. Forster, B. Richner, and R. Wattenhofer. Wireless Evacuation on m Rays with k Searchers. In *Proceedings of SIROCCO 2017, to appear*, 2017.

12 S. Brandt, J. Uitto, and R. Wattenhofer. A Tight Bound for Semi-Synchronous Collaborative Grid Exploration. In *32nd International Symposium on Distributed Computing (DISC)*, 2018.

13 P. Brass, F. Cabrera-Mora, A. Gasparri, and J. Xiao. Multirobot Tree and Graph Exploration. *IEEE Transactions on Robotics*, 27(4):707–717, August 2011. doi:10.1109/TRO.2011.2121170.

14 E. Castello, T. Yamamoto, F. d. Libera, W. Liu, A. Winfield, and Y. Nakamura. Adaptive foraging for simulated and real robotic swarms: the dynamical response threshold approach. *Swarm Intelligence*, 10(1):1–31, 2016.

15 S. Chopra and M. Egerstedt. Multi-Robot Routing for Servicing Spatio-Temporal Requests: A Musically Inspired Problem. In *IFAC Conference on analysis and design of hybrid systems*, 2012.

16 M. Chrobak, L. Gasieniec, T. Gorry, and R. Martin. Group Search on the Line. In *Proceedings of SOFSEM 2015: 41st International Conference on Current Trends in Theory and Practice of Computer Science*, pages 164–176, 2015.

17 L. Cohen, Emek Y, O. Louidor, and J. Uitto. Exploring an Infinite Space with Finite Memory Scouts. In *Proc. of the 28th SODA*, SODA '17, pages 207–224, 2017.

18 S. Das, P. Flocchini, G. Prencipe, N. Santoro, and M. Yamashita. Autonomous mobile robots with lights. *Theoretical Computer Science*, 609(P1):171–184, 2016.

19 X. Deng and C.H. Papadimitriou. Exploring an unknown graph (Extended Abstract). In *Proceedings of FOCS*, 1990.

20 Stefan Dobrev, Lata Narayanan, Jaroslav Opatrny, and Denis Pankratov. Exploration of High-Dimensional Grids by Finite State Machines. *Computing Research Repository (CoRR)*, abs/1902.03693, 2019. arXiv:1902.03693.

21 Y. Emek, T. Langner, D. Stolz, J. Uitto, and R. Wattenhofer. How many ants does it take to find the food? *Theoretical Computer Science*, 608:255–267, 2015.

22 Y. Emek, T. Langner, J. Uitto, and R. Wattenhofer. Solving the ANTS problem with finite state machines. In *Proceedings of ICALP*, pages 471–482, 2014.

23 M.A. Estrada, S. Mintchev, D. Christensen, M.R. Cutkosky, and D. Floreano. Forceful Manipulation with Micro Air Vehicles. *Science Robotics*, 2018.

24 O. Feinerman, A. Korman, Z. Lotker, and J-S. Sereni. Collaborative Search in the Plane without Communication. In *Proceedings of PODC*, pages 77–86, 2013.

25 P. Flocchini, G. Prencipe, and N. Santoro. *Distributed computing by oblivious mobile robots (Synthesis Lectures on Distributed Computing Theory)*. Morgan & Claypool Publishers, 2016.

26 P. Flocchini, G. Prencipe, N. Santoro, and P. Widmayer. Arbitrary pattern formation by asynchronous, anonymous, oblivious robots. *Theoretical Computer Science*, 407(1–3):412–447, 2008.

27 Paola Flocchini, David Ilcinkas, Andrzej Pelc, and Nicola Santoro. Remembering Without Memory: Tree Exploration by Asynchronous Oblivious Robots. *Theoretical Computer Science*, 411(14-15):1583–1598, March 2010. doi:10.1016/j.tcs.2010.01.007.

28 Pierre Fraigniaud, David Ilcinkas, Guy Peer, Andrzej Pelc, and David Peleg. Graph Exploration by a Finite Automaton. *Theoretical Computer Science*, 345(2-3):331–344, 2005. URL: https://hal.archives-ouvertes.fr/hal-00341531.

29 S. K. Ghosh and R. Klein. Online algorithms for searching and exploration in the plane. *Computer Science Review*, 4(4):189–201, 2010.

30 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

31 L. Hua and E. K. P. Chong. Search on lines and graphs. In *Proceedings of the 48th IEEE Conference on Decision and Control, CDC 2009, China*, pages 5780–5785, 2009.

32 A. Jez and J. Lopuszanski. On the two-dimensional cow search problem. *Information Processing Letters*, 109(11):543–547, 2009.

33 C. Lenzen, N. A. Lynch, C. C. Newport, and T. Radeva. Trade-offs between selection complexity and performance when searching the plane without communication. In *ACM Symposium on Principles of Distributed Comp. (PODC 2014),*, pages 252–261, 2014.

34 A. López-Ortiz and G. Sweet. Parallel searching on a lattice. In *Proceedings of the 13th Canadian Conference on Computational Geometry (CCCG 2001)*, pages 125–128, 2001.

35 Abraham Wald. On Cumulative Sums of Random Variables. *Ann. Math. Statist.*, 15(3):283–296, September 1944. doi:10.1214/aoms/1177731235.