The Power of Team Exploration: Two Robots Can Learn Unlabeled Directed Graphs

Michael A. Bender*
Aiken Computation Laboratory
Harvard University
Cambridge, MA 02138

Donna K. Slonim†
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

Abstract

We show that two cooperating robots can learn exactly any strongly-connected directed graph with n indistinguishable nodes in expected time polynomial in n. We introduce a new type of homing sequence for two robots which helps the robots recognize certain previously-seen nodes. We then present an algorithm in which the robots learn the graph and the homing sequence simultaneously by wandering actively through the graph. Unlike most previous learning results using homing sequences, our algorithm does not require a teacher to provide counterexamples. Furthermore, the algorithm can use efficiently any additional information available that distinguishes nodes. We also present an algorithm in which the robots learn by taking random walks. The rate at which a random walk converges to the stationary distribution is characterized by the conductance of the graph. Our random-walk algorithm learns in expected time polynomial in n and in the inverse of the conductance and is more efficient than the homing-sequence algorithm for high-conductance graphs.

1 Introduction

Consider a robot trying to construct a street map of an unfamiliar city by driving along the city's roads. Since many streets are one-way, the robot may be unable to retrace its steps. However, it can learn by using street signs to distinguish intersections. Now suppose that it is nighttime and that there are no street signs. The task becomes significantly more challenging.

In this paper we present a probabilistic polynomial-time algorithm to solve an abstraction of the above problem by using two cooperating learners. Instead of learning a city, we learn a strongly-connected directed graph G. Every node has d outgoing edges labeled from 0 to $d-1$. Nodes in the graph are indistinguishable, so a robot cannot recognize if it is placed on a node that it has previously seen. Moreover since the graph is directed, a robot is unable to retrace its steps while exploring. The robots can recognize when they are at the same node and can communicate freely by radio. Radio communication is used to synchronize actions. If we assume that the two robots move synchronously and share a polynomial-length random string, then no communication is necessary. Thus with only minor modifications, our algorithms may be used in a distributed setting.

Previous results showing the power of team learning are plentiful (*e.g.*, [CBF+93], [KS93], [DP+91]), particularly in the field of inductive inference (see Smith [Smi94] for an excellent survey). There are also many results on learning unknown graphs under various conditions (*e.g.*, [BRS93], [DP90], [RS87], [RS93]). Rabin proposed the idea of dropping pebbles to mark nodes [Rab67]. This suggestion led to work exploring the searching capabilities of a finite automaton supplied with pebbles (*e.g.*, [BS77], [BK78], [Sav72]). Cook and Rackoff generalized the idea of pebbles to jumping automata [CR80]. However, most previous work has concentrated on learning undirected graphs or graphs with distinguishable nodes.

The power behind the two-robot model lies in the robots' abilities to recognize each other and to move independently. Nonetheless, it is not obvious how to harness this power. If the robots separate in unknown territory, they could search for each other for an amount of time exponential in the size of the graph. Therefore, in any successful strategy for our model the two robots must always know how to find each other. One strategy that satisfies this requirement has both robots following the same path whenever they

*Author was supported by NSF Grant CCR-93-13775. Author's net address: bender@das.harvard.edu

†Author was supported by NSF Grant CCR-93-10888. Author's net address: slonim@theory.lcs.mit.edu

are in unmapped territory. They may travel at different speeds, however, with one robot scouting ahead and the other lagging behind. We call this a *lead-lag strategy*. In a lead-lag strategy the lagging robot must repeatedly make a difficult choice. The robot can wait at a particular node, thus marking it, but the leading robot may not find this marked node again in polynomial time. Alternatively, the lagging robot can abandon its current node to catch up with the leader, but then it may not know how to return to that node. In spite of these difficulties, our algorithms successfully employ a lead-lag strategy.

Our main algorithm runs without prior knowledge of the number of nodes in the graph, n, in time polynomial in n. We show that no probabilistic polynomial-time algorithm for a single robot with a constant number of pebbles can learn all unlabeled directed graphs when n is unknown. Thus, our algorithms demonstrate that two robots are strictly more powerful than one.

Rivest and Schapire [RS93] present an algorithm for a single robot to learn minimal deterministic finite automata. With the help of an equivalence oracle, their algorithm learns a homing sequence which it uses in place of a reset function. It then runs several copies of Angluin's algorithm [Ang87] for learning DFAs given a reset. Angluin has shown that any algorithm for learning DFAs by exploration requires an equivalence oracle [Ang81].

We introduce a new type of homing sequence for two robots. Because of the strength of the homing sequence, our algorithm does not require an equivalence oracle. For any graph, the expected running time of our algorithm is $O(d^2n^5)$. In practice, our algorithm can use additional information (*e.g.*, indegree, outdegree, or color of nodes) to find better homing sequences and to run faster.

Two robots can learn specific classes of directed graphs more quickly, such as the class of graphs with high conductance. Conductance, a measure of the expansion properties of a graph, was introduced by Sinclair and Jerrum [SJ89]. The class of directed graphs with high conductance includes graphs with exponentially-large cover time. We present a randomized algorithm which learns graphs with conductance greater than $n^{-\frac{1}{2}}$ in $O(dn^4 \ln^2 n)$ steps with high probability.

2 Preliminaries

Let $G = (V, E)$ represent the unknown graph, where G has n nodes, each with outdegree d. An edge from node u to node v with label i is denoted $\langle u, i, v\rangle$. We say that an algorithm *learns* graph G if it outputs a graph isomorphic to G. Our algorithms maintain a graph *map* which represents the subgraph of G learned so far. Included in *map* is an implicit start node u_0. A node u in *map* is called *unfinished* if it has any unexplored outgoing edges. Node u is *reachable* from node v if there is a path from v to u containing only edges in *map*. For robot k, the node in *map* corresponding to k's location in G if *map* is correct is denoted $Loc_M(k)$. Robot k's location in G is denoted $Loc_G(k)$.

Let f be an automorphism on the nodes of G such that

$$\forall a, b \in G, \langle a, i, b\rangle \in G \iff \langle f(a), i, f(b)\rangle \in G.$$

We say *nodes* c and d are *equivalent* (written $c \equiv d$) iff there exists such an f where $f(c) = d$.

We now present notation to describe the movements of k robots in a graph. An *action* A_i of the ith robot is either a label of an outgoing edge to explore, or the symbol r for "rest." A *k-robot sequence of actions* is a sequence of *steps* denoting the actions of the k robots; each step is a *k-tuple* $\langle A_0, \ldots, A_{k-1}\rangle$. For sequences s and t of actions, $s \circ t$ denotes the sequence of actions obtained by concatenating s and t.

A *path* is a sequence of edge labels. A robot *follows* a path by traversing the edges in the path in order beginning at a particular start node in *map*. The node in *map* reached by starting at u_0 and following *path* is denoted $final_M(path_c,\ u_0)$. Let s be a two-robot sequence of actions such that if both robots start together at any node in any graph and execute s, they follow exactly the same path, although perhaps at different speeds. We call such a sequence a *lead-lag* sequence. Note that if two robots start together and execute a lead-lag sequence, they end together. The node in G reached if both robots start at node a in G and follow lead-lag sequence s is denoted $final_G(s,\ a)$.

In all the procedures in this paper, the variables are passed by reference and are modified destructively. For convenience, we name our robots Lewis and Clark.

3 Reset

The problem of learning a directed graph with indistinguishable nodes is difficult because once both robots have left a known portion of the graph, they do not know how to return. This problem would vanish if there were a reset function that could transport both robots to a particular start node u_0. We describe an algorithm for two robots to learn directed graphs

```
Learn-with-Reset( ):
map := ({u_0}, ∅)                          { map is the graph consisting of node u_0 and no edges }
path := empty path                         { path is the null sequence of edge labels }
while there are unfinished nodes in map
    Learn-Edge(map,path)
    Reset
return map
```

```
Learn-Edge(map,path):
Lewis follows path to final_M(path)
u_i := some unfinished node in map reachable from Loc_M(Lewis)
Lewis moves to node u_i; append the path taken to path
pick an unexplored edge l out of node u_i
Lewis moves along edge l; append edge l to path          { Lewis crosses a new edge }
Clark follows path to final_M(path)                      { Clark looks for Lewis }
k := max_ℓ such that u_ℓ ∈ map                           { u_k is the most-recently learned vertex in map }
if ∃j ≤ k such that Clark first encountered Lewis at node u_j
    add ⟨u_i, l, u_j⟩ to map
else
    add new node u_{k+1} and ⟨u_i, l, u_{k+1}⟩ to map
```

given a reset. Having a reset is not a realistic model, but this algorithm forms the core of later algorithms which learn without a reset.

To learn a new edge in algorithm **Learn-with-Reset**, Lewis crosses the edge and then Clark tours the entire known portion of the map. If they encounter each other, Lewis's position is identified; otherwise Lewis is at a new node. The depth-first search strategy employed by **Learn-Edge** is essential in later algorithms.

Lemma 1 *The variable* path *in* **Learn-with-Reset** *denotes a tour of length* $\leq dn^2$ *that starts at* u_0 *and traverses all edges in map.*

Proof: Every time Lewis crosses an edge, that edge is appended to *path*. Since no edge is added to *map* until Lewis has crossed it, *path* must traverse all edges in *map*. In each call to **Learn-Edge**, at most n edges will be added to *path*. The body of the while loop is executed dn times, so $length(path) \leq dn^2$. □

Theorem 2 *After* $O(d^2n^3)$ *moves and* dn *calls to* **Reset**, **Learn-with-Reset** *halts and outputs a graph isomorphic to* G.

Proof sketch: By Lemma 1 and induction on the number of edges in *map*, *map* is always a subgraph of G. Furthermore, if *map* contains any unfinished nodes, then there is always some unfinished node in *map* reachable from $final_M(path)$. Otherwise, G would not be strongly connected. The correctness of the output follows. In the body of the while loop, each robot takes $length(path) \leq dn^2$ steps. The while loop is repeated at most dn times, so the algorithm halts within $O(d^2n^3)$ steps. □

4 Homing Sequences

In practice, robots learning a graph do not have access to a reset function. In this section we suggest an alternative technique: we introduce a new type of homing sequence for two robots.

Intuitively, a homing sequence is a sequence of actions whose observed output uniquely determines the final node reached in G. Rivest and Schapire [RS93] show how a single robot with a teacher can use homing sequences to learn strongly-connected minimal DFAs. The output at each node indicates whether that node is an accepting or rejecting state of the automaton. If the target DFA is not minimal, their algorithm learns the minimal encoding of the DFA. In other words, their algorithm learns the function that the graph computes rather than the structure of the graph.

In unlabeled graphs the nodes do not produce output. However, two robots can generate output indicating when they meet.

Definitions: Each step of a two-robot sequence of actions produces an output symbol T if the robots

```
Learn-with-HS(h):
done := FALSE
while not done
    execute h; c := the output sequence produced                 { instead of a reset }
    if map_c is undefined
        map_c := ({u_0}, ∅)        { map_c is the graph consisting of node u_0 and no edges }
        path_c := empty path               { path_c is the null sequence of edge labels }
    Learn-Edge(map_c,path_c)
    if map_c has no unfinished nodes
        done := TRUE
return map_c
```

are together and S if they are separate. An *output sequence* is a string in $\{T, S\}^*$ denoting the observed output of a sequence of actions.

Let s be a lead-lag sequence of actions and let a be a node in G. Then *output(s,a)* denotes the output produced by executing the sequence s, given that *both* robots start at a.

Definition: A lead-lag sequence s of actions is a *two-robot homing sequence* iff $\forall$ nodes $u, v \in G$,

$$output(s, u) = output(s, v) \Rightarrow final_G(s, u) \equiv final_G(s, v).$$

Because the output of a sequence depends on the positions of both robots, it provides information about the underlying structure of the graph. This new type of homing sequence is powerful. Unlike most previous learning results using homing sequences, our algorithms do not require a teacher to provide counterexamples.

In fact, two robots on a graph define a DFA whose states are pairs of nodes in G and whose edges correspond to pairs of actions. Since the automata defined in this way form a restricted class of DFAs, our results are compatible with Angluin's work [Ang81] showing that a teacher is necessary for learning general DFAs.

Given a strongly-connected graph G and a node a in G, a pair of robots can verify whether they are together at a node equivalent to a on some graph isomorphic to G. We describe a verification algorithm **Verify**(a, G) in Section 5.

Theorem 3 *Every strongly-connected directed graph has a two-robot homing sequence.*

Proof: The following algorithm (based on that of Kohavi [Koh78, RS93]) constructs a homing sequence: Initially, let h be empty. As long as there are two nodes u and v in G such that $output(h,u) = output(h,v)$ but $final(h,u) \not\equiv final(h,v)$, let x be a lead-lag sequence whose output distinguishes *final(h,u)* from *final(h,v)*. Since $final(h,u) \not\equiv final(h,v)$ and G is strongly connected, such an x always exists. Append x to h.

Each time a sequence is appended to h, the number of different outputs of h increases by at least 1. Since G has n nodes, there are at most n possible output sequences. Therefore, after $n-1$ iterations, h is a homing sequence. □

In Section 5 we show that it is possible to find a counterexample x efficiently. In fact, the sequence of actions returned by a call of **Verify**(u, G) is always a suitable x. Using the bound from Corollary 6, we claim that this algorithm produces a homing sequence of length $O(n^4)$ for all graphs. Note that in many cases, shorter homing sequences may exist.

4.1 Using a Homing Sequence to Learn

Given a homing sequence h, an algorithm can learn G by maintaining several running copies of **Learn-with-Reset**. Instead of a single start node, there are as many as n possible start nodes, each corresponding to a different output sequence of h. Note that many distinct output sequences may be associated with the same final node in G.

The new algorithm **Learn-with-HS** maintains several copies of *map* and *path*, one for each output sequence of h. Thus, graph map_c denotes the copy of the map associated with output sequence c. Initially, Lewis and Clark are at the same node. Whenever algorithm **Learn-with-Reset** would use a reset, **Learn-with-HS** executes the homing sequence h. If the output of h is c, the algorithm learns a new edge in map_c as if it had been reset to u_0 in map_c. After each execution of h, the algorithm learns a new edge in some map_c. Since there are at most n copies, each with kn edges to learn, eventually one map will be completed. Recall that a homing sequence is a lead-lag sequence. Therefore, at the beginning and end of every homing sequence the two robots are together.

Theorem 4 *If* **Learn-with-HS** *is called with a homing sequence h as input, it will halt after $O(d^2n^4 + dn^2|h|)$ steps and output a graph isomorphic to G.*

Proof sketch: The correctness of the algorithm follows from Theorem 2 and the definition of a two-robot homing sequence. Let $r = O(d^2n^3)$ be the number of steps taken by **Learn-with-Reset**. Since there are at most n start nodes, **Learn-with-HS** takes at most $nr + dn^2|h|$ steps. □

5 Learning the Homing Sequence

Unlike a reset function, a two-robot homing sequence can be learned. The algorithm **Learn-Graph** maintains a candidate homing sequence h and improves h as it learns G.

Definition: Candidate homing sequence h is called a *bad* homing sequence if there exist nodes u, v, $u \neq v$, such that $output(h, u) = output(h, v)$, but $final(h, u) \neq final(h, v)$.

Definition: Let a be a node in G. We say that map_c is a *good representation* of $\langle a, G\rangle$ iff there exists an isomorphism f from the nodes in $map_c = (V^c, E^c)$ to the nodes in a subgraph $G' = (V', E')$ of G, such that $f(u_0) = a$, and

$$\forall u_i, u_j \in V^c, \langle u_i, \ell, u_j\rangle \in E^c \Longleftrightarrow \langle f(u_i), \ell, f(u_j)\rangle \in E'.$$

In algorithms **Learn-with-Reset** and **Learn-with-HS**, the graphs *map* and map_c are *always* good representations of G. In **Learn-Graph** if the candidate homing sequence h is bad, a particular map_c may not be a good representation of G. However, the algorithm *can* test for such maps. Whenever a map_c is shown to be in error, h is improved and all maps are discarded. By the proof of Theorem 3, we know that a candidate homing sequence must be improved at most $n-1$ times. In Section 5.1 we explain how to use adaptive homing sequences to discard only one map per improvement.

We now define a test which with probability at least $\frac{1}{n}$ detects an error in map_c if one exists.

Definition: Let $path_c$ be a path such that a robot starting at u_0 and following $path_c$ traverses every edge in $map_c = (V^c, E^c)$. Let $u_0 \ldots u_m$ be the nodes in V^c numbered in order of their first appearance in $path_c$. If both robots are at u_0 then $test_c(u_i)$ denotes the following sequence of actions: (1) Lewis follows $path_c$ to the first occurrence of u_i; (2) Clark follows $path_c$ to the end; (3) Lewis catches up to Clark.

Definition: Given map_c and any lead-lag sequence t of actions, define $expected(t,\ map_c)$ to be the expected output if map_c is correct and if both robots start at node u_0 and execute sequence t. We abbreviate $expected(test_c(u_i), map_c)$ by $expected(test_c(u_i))$.

Lemma 5 *Suppose Lewis and Clark are both at some node a in G. Let* path_c *be a path such that a robot starting at u_0 and following* path_c *traverses every edge in* map_c*. Then* map_c *is a good representation of $\langle a, G\rangle$ iff $\forall u_i \in V^c$, $\mathrm{output}(\mathrm{test}_c(u_i)) = \mathrm{expected}(\mathrm{test}_c(u_i))$.*

Proof:
($\Longrightarrow$): By definition of good representation and $expected(\mathrm{test}_c(u_i))$.
($\Longleftarrow$): Suppose that all tests produce the expected output. We define a function f as follows: Let $f(s) = a$. Let $p(u_i)$ be the prefix of $path_c$ up to the first occurrence of u_i. Define $f(u_i)$ to be the node in G that a robot reaches if it starts at a and follows $p(u_i)$. Let $G' = (V', E')$ be the image of $f(map_c)$ on (V, E).

We first show that f is an isomorphism from V^c to V'. By definition of G', f must be surjective. To see that f is injective, assume the contrary. Then there exist two nodes $u_i, u_j \in V^c$ such that $i \neq j$ but $f(u_i) = f(u_j)$. But then $output(\mathrm{test}_c(u_i)) \neq expected(\mathrm{test}_c(u_i))$, which contradicts our assumption that all tests succeed.

Next, we show that $\langle u_i, \ell, u_j\rangle \in V^c \Longleftrightarrow \langle f(u_i), \ell, f(u_j)\rangle \in V'$, proving that map_c is a good representation of $\langle a, G\rangle$.
($\Longleftarrow$:) By definition of G', the image of map_c.
($\Longrightarrow$:) Inductively assume that $\langle u_i, \ell, u_j\rangle \in V^c \Longleftrightarrow \langle f(u_i), \ell, f(u_j)\rangle \in V'$ for the first m edges in $path_c$, and suppose that this prefix of the path visits only nodes $u_0 \ldots u_i$. Now consider the $(m+1)$st edge $e = \langle a, \ell, b\rangle$. There are two possibilities. In one case, edge e leads to some new node u_{i+1}. Then by definition $f(u_{i+1})$ is b's image in G, so $\langle f(a), \ell, f(b)\rangle \in G'$. Otherwise e leads to some previously-seen node u_{i-c}. Suppose that $f(u_{i-c})$ is *not* the node reached in G by starting at u_0 and following the first $m+1$ edges in $path_c$. Then $output(test_c(u_{i-k})) \neq expected(test_c(u_{i-k}))$, so $test_c(u_{i-k})$ fails, and we arrive at a contradiction. Therefore $f(b) = f(u_{i-c})$ and $\langle f(a), \ell, f(b)\rangle \in G'$. □

Learn-Graph( ):

```
done := FALSE
h := Λ (empty sequence)
while not done
    execute h; c := the output sequence produced                     { instead of a reset }
    if map_c is undefined
        map_c := ({u_0}, ∅)          { map_c is the graph consisting of node u_0 and no edges }
        path_c := empty path                  { path_c is the null sequence of edge labels }
    v := value of a fair 0/1 coin flip                 { learn edges or test for errors? }
    if map_c has no unfinished node reachable from final_M(path_c)
        Lewis and Clark move to final_M(path_c)
        comp := maximal strongly-connected component in map_c containing final_M(path_c)
        h-improve := Verify(final_M(path_c), comp)
        if h-improve = Λ
            done := TRUE                                             { map_c is complete }
        else                                             { h-improve ≠ Λ. error detected }
            append h-improve to end of h                       { improve homing sequence ...}
            discard all maps and paths              { ...and start learning maps from scratch }
    else if v = 0                                                        { test for errors }
        u_i := a random node in map_c                          { randomly pick node to test }
        h-improve := Test(map_c, path_c, i)
        if h-improve ≠ Λ                                                  { error detected }
            append h-improve to end of h                       { improve homing sequence ...}
            discard all maps and paths              { ...and start learning maps from scratch }
    else
        Learn-Edge(map_c, path_c)
return map_c
```

Test(map_c, $path_c$, i): $\{u_0, u_1, \ldots, u_k$ are the nodes in map_c indexed by first appearance in $path_c$ $\}$

```
h-improve := the following sequence of actions:
    Lewis follows path_c to the first occurrence of u_i in path_c
    Clark follows path_c to end
    Lewis follows path_c to catch up to Clark
if output(h-improve) ≠ expected-output(h-improve)                       { if error detected }
    return h-improve                                                    { return test_c(u_i) }
else
    return Λ                                                         { return empty sequence }
```

Verify(v_0, map): $\{ v_0, v_1, \ldots, v_k$ are the nodes in map ordered by first appearance in $p\}$

```
path := path such that a robot starting at v_0 in map and following path visits all nodes
    in map and returns to node
h-improve := Λ
foreach i, 0 ≤ i < k
    t := Test(map, path, i)
    if h-improve = Λ
        h-improve := t
return h-improve
```

Corollary 6 *Suppose Lewis and Clark are together at* u_0 *in* map_c. *Let* map_c *be strongly connected and have* n *nodes,* $u_0, \ldots u_{n-1}$. *Then the two robots can verify whether* map_c *is a good representation of* $\langle Loc_G(Lewis), G\rangle$ *in* $O(n^3)$ *steps.*

Proof: Since map_c is strongly connected, there exists a path $path_c$ with the following property: a robot starting at u_0 and following $path_c$ visits all nodes in map_c and returns to u_0. Index the remaining nodes in order of their first appearance in $path_c$. The two robots verify whether, for all u_i in order, $output(test_c(u_i)) = expected(test_c(u_i))$. Note that Lewis and Clark are together at u_0 after each test. By Lemma 5, this procedure verifies map_c. Since $path_c$ has length $O(n^2)$, each test has length $O(n^2)$, so verification requires at most $O(n^3)$ steps. □

In **Learn-Graph** after the robots execute a homing sequence, they randomly decide either to learn a new edge or to test a random node in map_c. The following lemma shows that a test that failed can be used to improve the homing sequence.

Lemma 7 *Let* h *be a candidate homing sequence in* **Learn-Graph**, *and let* u_k *be a node such that* $\text{output}(\text{test}_c(u_k)) \neq \text{expected}(\text{test}_c(u_k))$. *Then there are two nodes* a, b *in* G *that* h *does not distinguish but that* $h \circ \text{test}_c(u_k)$ *does.*

Proof: Let a be a node in G such that when both robots start at a, $output(test_c(u_k)) \neq expected(test_c(u_k))$. Suppose that at step i in $test_c(u_k)$, the expected output is T (respectively S), but the actual output is S (resp. T). Each edge in $path_c$ and map_c was learned using **Learn-Edge**. If map_c indicates that the ith node in $path_c$ is u_k, there must be a start node b in G where u_k really is the ith node in $path_c$. Since $output(test_c(u_k)) \neq expected(test_c(u_k))$, the sequence $h \circ test_c(u_k)$ distinguishes a from b. □

The algorithm **Learn-Graph** runs until there are no more reachable unexplored nodes in some map_c. If map_c is not strongly connected, then it is not a good representation of G. In this case, the representation of the last strongly-connected component on $path$ must be incorrect. Thus, calling **Verify** on this component from the last node on $path$ returns a sequence that improves h. If map_c is strongly connected, then either **Verify** returns an improvement to the homing sequence, or map_c is a good representation of G.

Theorem 8 *The algorithm* **Learn-Graph** *always outputs a map isomorphic to* G *and is expected to halt in* $O(d^2n^6)$ *steps.*

The proof of Theorem 8, which will appear in the full paper, relies on the preceding lemmas and Theorem 4.

5.1 Improvements to the Algorithm

The running time for **Learn-Graph** can be diminished significantly. For example, we can use two-robot *adaptive* homing sequences. As in Rivest and Schapire [RS93], an *adaptive homing sequence* is a decision tree, so the actions in later steps of the sequence depend on the output of earlier steps. With an adaptive homing sequence, only one map_c needs to be discarded each time the homing sequence is improved. Thus the running time of **Learn-Graph** decreases by a factor of n to $O(d^2n^5)$.

Any additional information that distinguishes nodes can be included in the output, so homing sequences can be shortened. For example, a robot learning an unfamiliar city could easily count the number of roads leading into and out of intersections. It might also recognize stop signs, traffic lights, McDonalds' restaurants, or other common landmarks. Therefore, in any practical application of this algorithm we expect a significantly lower running time than the $O(d^2n^5)$ bound suggests.

Graphs with high conductance can be learned even faster using the algorithm presented in Section 6.

5.2 Limitations of a Single Robot with Pebbles

We now compare the computational power of two robots to that of one robot with a constant number of pebbles. Note that although **Learn-Graph** runs in time polynomial in n, the algorithm requires no prior knowledge of n. We argue here that a single robot with a constant number of pebbles cannot efficiently learn strongly-connected directed graphs without prior knowledge of n.

As a tool we introduce a family $\mathcal{C} = \cup_n \mathcal{C}_n$ of graphs called *combination locks* [Moo56]. For a graph $C = (V, E)$ in $\mathcal{C}_n$ (the class of n-node combination locks), $V = \{u_0, u_1, \ldots, u_{n-1}\}$ and either $\langle u_i, 0, u_{i+1 \, mod \, n}\rangle$ and $\langle u_i, 1, u_0\rangle \in E$, or $\langle u_i, 1, u_{i+1 \, mod \, n}\rangle$ and $\langle u_i, 0, u_0\rangle \in E$, for all $i \leq n$ (see Figure 1a). In order for a robot to "pick a lock" in $\mathcal{C}_n$ — that is, to reach node u_{n-1} — it must follow the unique n-node simple path from u_0 to u_{n-1}. Thus any algorithm for a single robot with *no* pebbles can expect to take $\Theta(2^n)$ steps to pick a random combination lock in $\mathcal{C}_n$.

We construct a restricted family $\mathcal{R}$ of graphs and consider algorithms for a single robot with a single

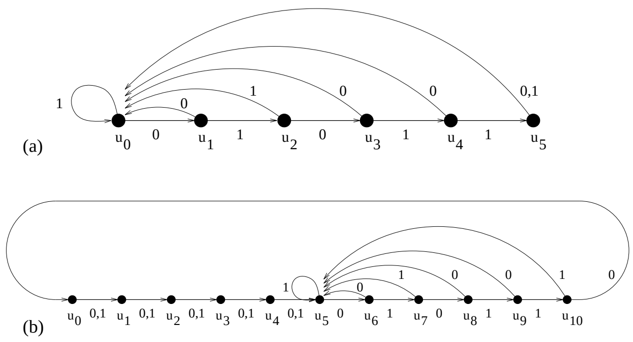

Figure 1: (a) A combination-lock, whose combination is $0-1-0-1-1$. (b) A graph in $\mathcal{R}_{11}$. Graphs in $\mathcal{R} = \cup_{n=1}^{\infty}\mathcal{R}_n$ cannot be learned by one robot with a constant number of pebbles.

pebble. For all positive integers n, the class $\mathcal{R}_n$ contains all graphs consisting of a directed ring of $n/2$ nodes with an $n/2$-node combination lock inserted into the ring (as in Figure 1b). Let $\mathcal{R} = \cup_{n=1}^{\infty}\mathcal{R}_n$. We claim that there is no probabilistic algorithm for one robot and one pebble that learns arbitrary graphs in $\mathcal{R}$ in polynomial time with high probability.

To see the claim, consider a single robot in node u_0 of a random graph in $\mathcal{R}$. Until the robot drops its pebble for the first time it has no information about the graph. Furthermore, with high probability the robot needs to take $\Theta(2^n)$ steps to emerge from a randomly-chosen n-node combination lock unless it drops a pebble in the lock. But since the size of the graph is unknown, the robot always risks dropping the pebble before entering the lock. If the pebble is dropped outside the lock, the robot will not see the pebble again until it has passed through the lock. A robot that cannot find its pebble has no way of marking nodes and cannot learn.

More formally, suppose that there were some probabilistic algorithm for one robot and a pebble to learn random graphs in $\mathcal{R}$ in polynomial time with probability greater than 1/2. Then there must be some constant c such that the probability that the robot drops its pebble in its first c steps is greater than 1/2. (Otherwise, the probability that the algorithm *fails* to learn in time polynomial in n is greater than 1/2.) Therefore, the probability that the robot loses its pebble and fails to learn a random graph in $\mathcal{R}_{2c}$ efficiently is at least 1/2. A similar argument holds for a robot with a constant number of pebbles. We conjecture that even if the algorithm is given n as input, a single robot with a constant number of pebbles cannot learn strongly-connected directed graphs. However, using techniques similar to those in Section 6, one robot with a constant number of pebbles and prior knowledge of n *can* learn high-conductance graphs in polynomial time with high probability.

6 Learning High Conductance Graphs

For graphs with good expansion, learning by walking randomly is more efficient than by using homing sequences. In this section we define conductance and present an algorithm which runs more quickly than **Learn-Graph** for graphs with conductance greater than $\frac{1}{n}$.

6.1 Conductance

The conductance [SJ89] of an ergodic random walk on a graph characterizes the rate at which the walk converges to the stationary distribution π. For a given directed graph $G = (V, E)$, consider a weighted graph $G' = (V, E, W)$ with the same vertices and edges as G, but with edge weights defined as follows. Let $M = \{m_{i,j}\}$ be the transition matrix of a random walk that leaves i by each outgoing edge with probability $1/(2 \cdot \text{degree}(i))$ and remains at node i with probability $\frac{1}{2}$. Let P^0 be an initial distribution on the n nodes in G, and let $P^t = P^0 M^t$ be the distribution after t steps of the walk defined by M. (Note that π is a *steady state* distribution if for every node i, $P_i^t = \pi_i \longrightarrow P_i^{t+1} = \pi_i$. For irreducible and aperiodic Markov chains, π exists and is unique.) Then the edge weight $w_{i,j} = \pi_i m_{i,j}$ is proportional to the steady state probability of traversing the edge from i to j. Note that the total weight entering a node is equal to the total weight leaving it; that is, $\sum_j w_{i,j} = \sum_j w_{j,i}$.

Consider a set $S \subseteq V$ which defines a cut $(S, \overline{S})$. For sets of nodes S and T, let $W_{S,T} = \sum_{s\in S, t\in T} w_{s,t}$. We denote $W_{S,V}$ by W_S, so W_V represents the total weight of all edges in the graph. Then the conductance of S is defined as $\phi_S = W_{S,\overline{S}}/\sum_{i\in S}\pi_i = W_{S,\overline{S}}/W_S$.

The conductance of a graph is the least conductance over all cuts whose total weight is at most $\frac{W_V}{2}$: $\phi(G) = \min_S \left\{\max(\phi_S, \phi_{\overline{S}})\right\}$. The conductance of a directed graph can be exponentially small.

Mihail [Mih89] shows that after a walk of length $\phi^{-2}\log\frac{2n}{\epsilon^2}$, the L_1 norm of the distance between the current distribution P and the stationary distribution π is at most ϵ (i.e. $\sum_i |P_i - \pi_i| \leq \epsilon$). In the rest of this section, a choice of $\epsilon = \frac{1}{n^2}$ is sufficient, so a random walk of length $\phi^{-2}\log 2n^5$ is used to approximate the stationary distribution. We call $T = \phi^{-2}\log 2n^5$ the *approximate mixing time* of a random walk on an n-node graph with conductance ϕ.

Learn-Graph2( ):

1 *done* := FALSE
2 T := $\phi^{-2} \log (2n^5)$ { the mixing time }
3 **while** (not (*done*))
4 $\quad$ *map* := $(\{u_0\}, \emptyset)$ { *map* is the graph consisting of node u_0 and no edges }
5 $\quad$ *lost* := FALSE
6 $\quad$ Lewis and Clark together take a random walk of length T
7 $\quad$ Lewis takes w random walks of length T { approx. stationary prob. of Loc_G(Clark) }
8 $\quad\quad$ x := number of walks where Lewis and Clark are together at the last step
9 $\quad$ **if** $\frac{x}{w} \leq \frac{3}{4n}$
10 $\quad\quad$ Clark follows *path* to catch up to Lewis { not at a frequently-visited node }
11 $\quad$ **else**
12 $\quad\quad$ Lewis moves randomly until he sees Clark { call the node where they meet u_0 }
13 $\quad\quad$ *done* := **Build-Map**(*map*)
14 **return** *map*

6.2 An Algorithm for High Conductance Graphs

If G has high conductance, we can use this additional information to learn the graph more quickly. In a high-conductance graph, the robots can estimate the steady state probability of node i by leaving Clark at node i while Lewis takes w random walks of $\phi^{-2} \log 2n^5$ steps each. Let x be the number of times that Lewis sees Clark at the last step of a walk. If w is large enough, $\frac{x}{w}$ is a good approximation to π_i.

Definitions: Call a node i a *likely* node if $\pi_i \geq \frac{1}{2n}$. Note that every graph must have at least one such node. A node that is not likely is called *unlikely.*

Algorithm **Learn-Graph2** uses the estimation technique described above to find a likely node u_0 and then calls the procedure **Build-Map** to construct a map of G starting from u_0.

Build-Map learns a new edge each iteration by sending Lewis across an unexplored edge $\langle u, \ell, v \rangle$ of some unfinished node u in *map*. Clark waits at start node u_0 while Lewis walks randomly until he meets Clark. (If u_0 is a likely node, this walk is expected to take $O(Tn)$ moves.) Lewis stores this random walk in the variable *path*. The procedure **Compress-Path** returns the shortest subpath of *path* that connects v to u_0. Finally, **Truncate-Path-at-Map** compares nodes on the path with all nodes in *map* and returns the shortest subpath connecting v to some node in *map*. By adding the final path to the map, **Build-Map** connects the new node v to *map*, so *map* always contains a strongly connected subgraph of G.

The proof of the following theorem will appear in the full paper. The proof of *(1)* follows easily from the strong connectivity of G, while the proof of *(2)* is long but fairly straightforward. It relies on efficient implementations of **Compress-Path** and **Truncate-Path-at-Map** and on the repeated application of Chernoff bounds.

Theorem 9 *Suppose* **Learn-Graph2** *is run on a graph G with $w = 6\alpha n \ln n$ for some constant α. Then (1) when* **Learn-Graph2** *halts, it outputs a graph isomorphic to G, and (2) with probability at least $1 - 2dn^{-\frac{\alpha}{2}+1}$ it halts within $O(dT\alpha n^3 \ln n)$ steps.*

6.2.1 Exploring Without Prior Knowledge

Prior knowledge of n is used in two ways in **Learn-Graph2**: to estimate the stationary probability of a node and to compute the mixing time T. If T is known but n is not, the algorithm can forego estimating π_i entirely and simply run **Build-Map** after step 6. The removal of lines 7 – 12 from **Learn-Graph2** yields a new algorithm whose running time is slower than the original **Learn-Graph2** by a factor of $O(n \ln n)$. In this new algorithm, standard doubling can be used to estimate the mixing time T. Thus no prior knowledge of the graph is necessary.

7 Conclusions and Open Problems

Note that a single robot with a pebble can simulate algorithm **Learn-Graph2** with a substantial slowdown. However, **Learn-Graph2** does not run in polynomial expected time on graphs with exponentially-small conductance. An open problem is to establish

```
Build-Map( map ):
while there are unfinished nodes in map and not lost        { Lewis and Clark both at u_0 }
    u_i := an unfinished node in map
    ℓ := an unexplored edge out of u_i
    Lewis moves to u_i and crosses unexplored edge ℓ        { Lewis crosses a new edge }
    path := empty path
    while length(path) < Tαn ln n and robots are not together        { Lewis walks randomly ...}
        Lewis traverses a random edge ℓ' and adds ℓ' to path
    if robots are together        { ...until he sees Clark at u_0 }
        both robots go to u_i and cross edge ℓ
        restart := a minimal path in map from u_0 to u_i and across edge ℓ
        path := Compress-Path(path, restart)        { removes loops from path }
        Lewis follows restart from u_0
        path, u_j := Truncate-Path-At-Map(path, map)        { shortest path back to map }
        add all nodes and edges in path from u_i to u_j to map
        both robots move to u_0
    else        { if Lewis walks Tαn ln n steps without seeing Clark }
        Clark follows path to catch up to Lewis
        lost := TRUE
if lost return FALSE
else return TRUE
```

tight bounds on the running time of an algorithm that uses one robot and a constant number of pebbles to learn an n-node graph G. We conjecture that the lower bound will be a function of $\phi(G)$, but there may be other graph characteristics (*e.g.*, cover time) which yield better bounds. It would also be interesting to establish tight bounds on the number of pebbles a single robot needs to learn graphs in polynomial time.

Another direction for future work is to find other special classes of graphs that two robots can learn substantially more quickly than general directed graphs, and to find efficient algorithms for these cases.

Acknowledgements

We gratefully thank Michael Rabin, Ron Rivest, Mike Sipser and Les Valiant for many enjoyable and helpful discussions. We also thank Rob Schapire for his insightful comments on an earlier draft of the paper.

References

[Ang81] Dana Angluin. A note on the number of queries needed to identify regular languages. *Information and Control*, 51:76–87, 1981.

[Ang87] Dana Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, November 1987.

[BK78] Manuel Blum and Dexter Kozen. On the power of the compass (or, why mazes are easier to search than graphs). In *19th Annual Symposium on Foundations of Computer Science*, pages 132–142. IEEE, 1978.

[BRS93] Margrit Betke, Ronald Rivest, and Mona Singh. Piecemeal learning of an unknown environment. In *Proceedings of the 1993 Conference on Computational Learning Theory*, pages 277–286, Santa Cruz, CA, July 1993. (Published as MIT AI-Memo 1474, CBCL-Memo 93; to be published in Machine Learning).

[BS77] M. Blum and W. J. Sakoda. On the capability of finite automata in 2 and 3 dimensional space. In *18th Annual Symposium on Foundations of Computer Science*, pages 147–161. IEEE, 1977.

[CBF+93] N. Cesa-Bianchi, Y. Freund, D. Helmbold, D. Haussler, R. Schapire, and M. Warmuth. How to use expert advice. In *Proceedings of the Eighteenth Annual ACM*

Symposium on Theory of Computing, pages 382–391, San Diego, CA, May 1993.

[CR80] Stephen A. Cook and Charles W. Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM Journal on Computing*, 9:636–652, 1980.

[DP90] Xiaotie Deng and Christos H. Papadimitriou. Exploring an unknown graph. In *Proceedings of the 31st Symposium on Foundations of Computer Science*, volume I, pages 355–361, 1990.

[DP+91] Robert Daley, Leonard Pitt, Mahendran Velauthapillai, and Todd Will. Relations between probabilistic and team one-shot learners. In *Proceedings of COLT '91*, pages 228–239. Morgan Kaufmann, 1991.

[Koh78] Zvi Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, second edition, 1978.

[KS93] Michael J. Kearns and H. Sebastian Seung. Learning from a population of hypotheses. In *Proceedings of COLT '93*, pages 101–110, 1993.

[Mih89] Milena Mihail. Conductance and convergence of Markov chains – a combinatorial treatment of expanders. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science*, pages 526–531, 1989.

[Moo56] Edward F. Moore. *Gedanken-Experiments on Sequential Machines*, pages 129–153. Princeton University Press, 1956. Edited by C. E. Shannon and J. McCarthy.

[Rab67] Michael O. Rabin. Maze threading automata. October 1967. Seminar talk presented at the University of California at Berkeley.

[RS87] Ronald L. Rivest and Robert E. Schapire. Diversity-based inference of finite automata. In *Proceedings of the Twenty-Eighth Annual Symposium on Foundations of Computer Science*, pages 78–87, Los Angeles, California, October 1987.

[RS93] Ronald L. Rivest and Robert E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, April 1993.

[Sav72] Walter J. Savitch. Maze recognizing automata and nondeterministic tape complexity. *JCSS*, 7:389–403, 1972.

[SJ89] Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82(1):93–133, July 1989.

[Smi94] Carl H. Smith. Three decades of team learning. To appear in *Proceedings of the Fourth International Workshop on on Algorithmic Learning Theory*. Springer Verlag, 1994.