Time-Space Tradeoffs for Undirected Graph Traversal by Graph Automata [*]

Paul Beame [†] Allan Borodin [‡] Prabhakar Raghavan [§] Walter L. Ruzzo [†]
Martin Tompa [†]

May 10, 1997

Abstract

We investigate time-space tradeoffs for traversing undirected graphs, using a variety of structured models that are all variants of Cook and Rackoff's "Jumping Automata for Graphs". Our strongest tradeoff is a quadratic lower bound on the product of time and space for graph traversal. For example, achieving linear time requires linear space, implying that depth-first search is optimal. Since our bound in fact applies to nondeterministic algorithms for *non*connectivity, it also implies that closure under complementation of nondeterministic space-bounded complexity classes is achieved only at the expense of increased time. To demonstrate that these structured models are realistic, we also investigate their power. In addition to admitting well known algorithms such as depth-first search and random walk, we show that one simple variant of this model is nearly as powerful as a Turing machine. Specifically, for general undirected graph problems, it can simulate a Turing machine with only a constant factor increase in space and a polynomial factor increase in time.

[*]This material is based upon work supported in part by the Natural Sciences and Engineering Research Council of Canada, by the National Science Foundation under Grants CCR-8703196, CCR-8858799, CCR-8907960, CCR-9002891, and CCR-9301186, and by IBM under Research Contract 16980043. A portion of this work was performed while the fourth author was visiting the University of Toronto, whose hospitality is gratefully acknowledged.

[†]Department of Computer Science and Engineering, University of Washington, Box 352350, Seattle, WA, U.S.A. 98195

[‡]Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 1A4

[§]IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, CA, U.S.A.. 95120

1. The Complexity of Graph Traversal

Graph traversal is a fundamental problem in computing, since it is the natural abstraction of many search processes, with applications as diverse as Internet searching (Mauldin and Leavitt [40], Selberg and Etzioni [46]) and computer-aided verification (Dill, *et al.* [25], Kurshan [38]). In computational complexity theory, graph traversal (or more precisely, st-connectivity) is a fundamental problem for an additional reason: understanding the complexity of directed versus undirected graph traversal seems to be the key to understanding the relationships among deterministic, probabilistic, and nondeterministic space-bounded algorithms. For instance, although directed graphs can be traversed nondeterministically in polynomial time and logarithmic space simultaneously, it is not widely believed that they can be traversed deterministically in polynomial time and small space simultaneously. (See Tompa [48] and Edmonds and Poon [27] for lower bounds, and Barnes *et al.* [5] for an upper bound.) In contrast, *undirected* graphs can be traversed in polynomial time and logarithmic space *probabilistically* by using a random walk (Aleliunas *et al.* [2], Borodin *et al.* [17]); this implies similar resource bounds on (nonuniform) deterministic algorithms (Aleliunas *et al.* [2]). More recent work presents uniform deterministic polynomial time algorithms for the undirected case using sublinear space (Barnes and Ruzzo [8]), and even $O(\log^2 n)$ space (Nisan [41]), as well as a deterministic algorithm using $O(\log^{1.5} n)$ space, but more than polynomial time (Nisan *et al.* [42]).

In this paper we concentrate on the undirected case. The simultaneous time and space requirements of the best known algorithms for undirected graph traversal are as follows. Depth-first or breadth-first search can traverse any n vertex, m edge undirected graph in $O(m+n)$ time, but requires $\Omega(n)$ space. Alternatively, a random walk can traverse an undirected graph using only $O(\log n)$ space, but requires $\Theta(mn)$ expected time (Aleliunas *et al.* [2]). In fact, Feige [28], based on earlier work of Broder *et al.* [20] and Barnes and Feige [7], has shown that there is a spectrum of compromises between time and space for this problem: any graph can be traversed in space S and expected time T, where $ST \leq mn(\log n)^{O(1)}/d_{min}$ and d_{min} is the minimum degree of any vertex. This raises the intriguing prospect of proving that logarithmic space and linear time are not simultaneously achievable or, more generally, proving a time-space tradeoff that closely matches these upper bounds.

Although it would be desirable to show a tradeoff for a general model of computation such as a random access machine, obtaining such a tradeoff is beyond the reach of current techniques. Thus it is natural to consider a "structured" model (Borodin [16]), that is, one whose basic move is based on the adjacencies of the graph, as opposed to one whose basic move is based on the bits in the graph's encoding. An appropriate structured model for proving such a tradeoff is some variant of the JAG ("jumping automaton for graphs") of Cook and Rackoff [24]. Such an automaton has a set of states, and a limited supply of pebbles that it can move from vertex to adjacent vertex ("walk") or directly to a vertex containing another pebble ("jump"). The purpose of its pebbles is to mark certain vertices temporarily, so that they are recognizable when some other pebble reaches them. The pebbles represent vertex names that a structured algorithm might record in its workspace. Walking represents replacing a vertex name by some adjacent vertex found in the input. Jumping represents copying a previously recorded vertex name.

Rabin (see [24]), Savitch [45], Blum and Sakoda [13], Blum and Kozen [12], Hemmerling [30] and others have considered similar models; see Hemmerling's monograph for an extensive bibliography (going back over a century) emphasizing results for "labyrinths" — graphs embedded in two- or three-dimensional Euclidean space.

The JAG is a structured model, but not a weak one. In particular, it is general enough to encompass in a natural way most known algorithms for undirected graph traversal. For instance, a JAG can execute a depth-first or breadth-first search, provided it has one pebble for each vertex, by leaving a pebble on each visited vertex in order to avoid revisiting it, and keeping the stack or queue of pebble names in its state. Furthermore, as Savitch [45] shows, a JAG with the additional power to move a pebble from vertex i to vertex $i + 1$ can simulate an arbitrary Turing machine on directed graphs. Even without this extra feature, we will show in Section 3 that JAGs are as powerful as Turing machines for the purposes of solving undirected graph problems (our main focus). In particular, we will show for all space bounds $S(n) = \Omega(\log n)$, that JAGs can solve any graph problem solvable by Turing machines in space $S(n)$, with at most a constant factor loss in space and a polynomial factor loss in time. Furthermore, the simulation requires only two pebbles and no jumping.

Cook and Rackoff define the time T used by a JAG to be the number of pebble moves, and the space to be $S = P \log_2 n + \log_2 Q$, where P is the number of pebbles and Q the number of states of the automaton. (Keeping track of the location of each pebble requires $\log_2 n$ bits of memory, and keeping track of the state requires $\log_2 Q$.) It is well known that st-connectivity for directed graphs can be solved by a deterministic Turing machine in $O(\log^2 n)$ space, by applying Savitch's Theorem [44] to the obvious $O(\log n)$ space nondeterministic algorithm for the problem. Cook and Rackoff show that the same $O(\log^2 n)$ space upper bound holds for deterministic JAGs by direct construction of an $O(\log n)$ pebble, $n^{O(1)}$ state deterministic JAG for directed st-connectivity. More interestingly, they also prove a lower bound of $\Omega(\log^2 n / \log \log n)$ on the space required by JAGs solving this problem, nearly matching the upper bound. Standard techniques (Adleman [1], Aleliunas *et al.* [2]) extend this result to any randomized JAG whose time bound is at most exponential in its space bound. Berman and Simon [11] extend this space lower bound to probabilistic JAGs with even larger time bounds, namely exponential in $(\log n)^{O(1)}$.

In this paper we use variants of the JAG to study the tradeoff between time and space for the problem of *undirected* graph traversal. The JAG variants we consider are in some ways more restricted than the model introduced by Cook and Rackoff, but in other ways are sometimes more powerful. For example, the variant studied in Section 4 is more restricted in its jumping ability, but is considerably more powerful in another dimension, namely, it is nondeterministic.

Several authors have considered traversal of undirected regular graphs by a JAG with an unlimited number of states but only the minimum number (one) of pebbles, a model better known as a *universal traversal sequence* (Aleliunas *et al.* [2], Alon *et al.* [3], Bar-Noy *et al.* [4], Borodin, Ruzzo, and Tompa [18], Bridgland [19], Buss and Tompa [21], Istrail [34], Karloff *et al.* [37], Tompa [49]). A result of Borodin, Ruzzo, and Tompa [18] shows that such an automaton requires $\Omega(m^2)$ time (on regular graphs with $3n/2 \leq m \leq n^2/6 - n$). Thus, for the particularly weak version of logarithmic space corresponding to the case $P = 1$, a quadratic lower bound on time is known.

The known algorithms and the lower bounds for universal traversal sequences suggest that the true time-space product for undirected graph traversal is approximately quadratic, perhaps $\Theta(mn)$. The main results of this paper are lower bounds for variants of the JAG that provide progress toward proving this conjecture and, in fact, establish such a lower bound for one variant.

These results are outlined below.

The upper bound of $ST \leq mn(\log n)^{O(1)}/d_{min}$ by Feige [28], and the preceding upper bounds of Broder *et al.* [20] and Barnes and Feige [7], are established on a model that is actually a restricted variant of the JAG. In their algorithms, the JAG initially drops $P-1$ pebbles on random vertices, after which they are never moved. It then uses its last pebble to explore the graph (probabilistically), with the others as fixed landmarks. In Section 4, using essentially the same variant of the JAG, we prove lower bounds of $PT = \Omega(mn/d) = \Omega(n^2)$ for d-regular graphs ($d \geq 3$), and $PT = \Omega(mn)$ for nonregular graphs, independent of the value of Q, even for nondeterministic JAGs. This nearly matches the upper bound. The main difference between the upper and lower bounds is that they are for complementary problems. The upper bound is by a one-sided error probabilistic algorithm for undirected st-connectivity. The lower bound applies to nondeterministic, and hence one-sided error probabilistic, algorithms for st-*non*connectivity. This result does not imply that Feige's algorithm is optimal, but does imply, for example, that it cannot be made both errorless (i.e., zero-sided error) and substantially faster (on this JAG model). It also implies optimality of depth- and breadth-first search, in the following sense. While it is not surprising that linear time is necessary for deciding connectivity (e.g., see Theorem 3), our quadratic lower bound shows the stronger result that achieving linear time requires linear space.

This result also bears on the complexity of undirected st-connectivity, versus that of its complement, st-nonconnectivity. For deterministic, or errorless probabilistic algorithms, of course, the two problems are of equal complexity. For nondeterministic or one-sided error probabilistic algorithms, however, the complexities may differ. In particular, if a problem L is solvable nondeterministically in $O(\log n)$ space, then the complement of L is, too, by the result of Immerman [33] and Szelepcsényi [47]. (For the problem of undirected st-connectivity, this also follows from the result of Nisan and Ta-Shma [43].) However, their algorithms are rather slow. For example, a logarithmic space nondeterministic RAM can solve st-connectivity in time $O(n)$, but to solve the complementary st-nonconnectivity problem by the Immerman or Szelepcsényi algorithms requires time $\Omega(n^4)$. Is nonconnectivity intrinsically more difficult? One of our results shows that this is indeed the case, at least on one class of structured models we consider. Namely, although both problems are solvable by a logarithmic space, polynomial time nondeterministic JAG, st-nonconnectivity is provably harder. Specifically, st-connectivity is solvable in $O(n)$ time by a logarithmic space nondeterministic JAG with only one pebble, a constant number of states, and no jumping. In contrast, we show that st-nonconnectivity requires more time, even on a somewhat richer model. Namely, time $\Omega(mn)$ ($\Omega(n^2)$ for regular graphs) is required to solve st-nonconnectivity by a nondeterministic JAG with one movable pebble and any fixed number of unmovable pebbles, even using exponentially many states and jumping.

The result above is the desired quadratic lower bound, on a model that is natural but more restricted than we would like. In particular, it would be nice to extend the result to a model in which all pebbles are movable. In fact, our proof does extend to give a nonlinear lower bound when some motion of the pebbles is allowed, but the bound degenerates when the pebbles are allowed to move with complete freedom. Such models are surprisingly powerful; see Section 3. Nevertheless, in a companion paper [9] we prove a lower bound on a model with freely moving pebbles, but without the ability to jump one pebble to another. This nonjumping model is closer to the one studied by Blum and Sakoda [13], Blum and Kozen [12] and Hemmerling [30]. We will distinguish this nonjumping variant by referring to it as a WAG — "walking automaton for graphs".

Following the preliminary appearance of some of these results [10], Edmonds [26] proved a much stronger result for traversing undirected graphs than that proved in [9], and Barnes and Edmonds [6] and Edmonds and Poon [27] proved even more dramatic tradeoffs for traversing directed graphs.

The results described above have the strength that they hold independent of the magnitude of Q, the number of states. Presumably the bounds can be strengthened by also accounting for Q. It is tempting to tackle first the case in which Q is constant; indeed, Cook and Rackoff [24] investigate JAGs on undirected graphs in this case, showing for example that $PQ = O(1)$ is impossible. For a nonjumping variant of JAGs, in Section 5 we prove the stronger bound $PQ = \Omega(n)$ for 2-regular graphs, no matter how much time the automaton is allowed. Thus, for logarithmic space, lower bounds on time are only interesting when the number of states grows at least linearly with the size of the graph. As one simple consequence, this makes the lower bounds harder to prove, as one cannot simply make the graph so large compared to Q that the automaton is guaranteed to loop forever among some states. As a byproduct, we show that a universal traversal sequence for 2-regular graphs cannot consist solely of the repetition of a short sequence.

Sections 3, 4, and 5 are largely self-contained, and may be read in any order.

2. Graph-Traversing Automata

The problem we will be considering is "undirected st-connectivity": given an undirected graph G and two distinguished vertices s and t, determine if there is a path from s to t.

Consider the set of all n-vertex, edge-labeled, undirected graphs $G = (V, E)$ with maximum degree d. For this definition, edges are labeled as follows. For every edge $\{u, v\} \in E$ there are two labels $\lambda_{u,v}, \lambda_{v,u} \in \{0, 1, \ldots, d-1\}$ with the property that, for every pair of distinct edges $\{u, v\}$ and $\{u, w\}$, $\lambda_{u,v} \neq \lambda_{u,w}$. It will sometimes be convenient to treat an undirected edge as a pair of directed *half edges*, each labeled by a single label. For example, the half edge directed from u to v is labeled $\lambda_{u,v}$.

We will also consider more restricted labelings, since this technical detail influences some of our results in unexpected ways. The general definition above requires that the outgoing labels from each vertex u be distinct. That is, for all u and all neighbors $v \neq v'$ of u we require $\lambda_{u,v} \neq \lambda_{u,v'}$. We define a graph to be *bijectively labeled* if, in addition, the incoming labels are distinct, i.e., $\lambda_{v,u} \neq \lambda_{v',u}$. Any graph G can be given a bijective labeling as follows. Form a bipartite graph on two copies of G's vertex set by adding a directed edge from u in the first copy to v in the second for each directed half edge from u to v in G. Any bipartite graph of maximum degree d can be d-edge-colored using matching techniques (see Bondy and Murty [15, Theorem 6.1]). The color of the edge (u, v) becomes the label $\lambda_{u,v}$ of the corresponding half edge in G.

A special case of bijective labelings are the *symmetric* labelings, where all edges have the same label in each direction, i.e., $\lambda_{u,v} = \lambda_{v,u}$ for all u, v. (Universal traversal sequences for regular graphs with bijective and symmetric labelings have been considered previously by Hoory and Wigderson [32] and Istrail [35], respectively, although under different names. Both papers used the term "consistent" for these two different classes of restricted labelings.)

Not all graphs have symmetric labelings, and while every graph does have a bijective labeling, such labelings are not known to be computable in logarithmic space. Nevertheless, Lemma 1 below shows that, when considering upper bounds for st-connectivity, there is no loss of generality in restricting attention to symmetrically (and hence, bijectively) labeled graphs. Of course, lower bounds are at least as strong if they also hold when restricted to such graphs.

The reduction mentioned in Lemma 1 is not intended to be implemented on a JAG, but rather on a general model of computation such as a Turing machine.

Lemma 1: There is a simple, connectivity-preserving, logarithmic space reduction from general labeled graphs to symmetrically labeled graphs of maximum degree at most three.

Proof: Let $r_{u,v}$ be the rank of $\lambda_{u,v}$ in $\{\lambda_{u,v'} \mid v' \text{ is adjacent to } u\}$. For example, if the graph is regular, $r_{u,v}$ is simply $\lambda_{u,v}$. Replace each vertex of degree d by a d-cycle, if d is even, and by a $(d+1)$-cycle, if d is odd. (For the purposes of this proof, a 2-cycle is simply an edge.) Label these cycles symmetrically using 0 and 1. Replace edge $\{u, v\}$ by an edge from the $r_{u,v}$-th vertex of u's cycle to the $r_{v,u}$-th vertex of v's cycle, symmetrically labeled 2. $\square$

It is not difficult to extend the proof to make the graph in Lemma 1 both symmetrically labeled and 3-regular.

Following Cook and Rackoff [24], a *JAG* is an automaton with Q states and P distinguishable pebbles, where both P and Q may depend on n and d. For the st-connectivity problem, two vertices s and t of its input graph are distinguished. The P pebbles are initially placed on s. Each move of the JAG depends on the current state, which pebbles coincide on vertices, which pebbles are on t, and the edge labels emanating from the pebbled vertices. Based on this information, the automaton changes state, and selects some pebble p and either some $i \in \{0, 1, \ldots, d-1\}$ or some $j \in \{1, 2, \ldots, P\}$. In the former case, i must be an edge label emanating from the vertex currently pebbled by p, and p is moved to the other endpoint of the edge with label i; in the latter case, p "jumps" to the vertex occupied by pebble j. (The decision to make t "visible" to the JAG but s "invisible" was made simply to render 1-pebble JAGs on regular graphs equivalent to universal traversal sequences.) A deterministic JAG that determines st-connectivity is required to enter an accepting state if and only if there is a path from s to t. Nondeterministic and probabilistic generalizations of JAGs are defined in the usual way. Note that JAGs are nonuniform models.

There are a number of interesting variants of JAGs. For instance, in Section 4 we will consider a strengthened form of jumping, called "strong jumping," where the automaton's move may also be to select some $v \in \{1, \ldots, n\}$ and jump pebble p to vertex v. On the other hand, in Section 5 we will disallow jumping by studying WAGs. We will also distinguish among three types of pebbles: "active", "passive", and "unmovable". The automaton as described in the previous paragraph has active pebbles, in the sense that any pebble can move. A weaker notion is that of the passive pebble, which cannot move unless accompanied by an active pebble. In this case, we allow one active pebble accompanied by any number of passive pebbles to walk or jump each move. Of particular interest is the case of one active pebble and $P-1$ passive pebbles, in which case it is natural to think of the automaton itself as the active pebble moving about the graph, picking up and dropping pebbles. This is the model used in Section 5.

Closely related to the passive pebble is the unmovable pebble, which, once placed on the graph, cannot be moved at all. This is the model discussed in Section 4. We will mainly consider unmovable pebbles as a special case of passive pebbles. That is, the automaton starts with a supply of pebbles that are carried and dropped at will (but never picked up). In Section 4.2, however, we will also consider a less uniform placement method where some of the pebbles are placed on the graph before the JAG begins its computation. Detailed definitions of this version are deferred to Section 4.2.

We have defined JAGs running on arbitrary graphs, but JAGs that are guaranteed to operate correctly only on regular graphs are also of interest, and sometimes may be substantially more efficient than in the general case; see Theorem 12, for example. Our lower bounds generally apply even to JAGs operating on regular graphs. The restriction to regular graphs, in addition to strengthening the lower bound results, provides comparability to the known results about universal traversal sequences. A technicality that must be considered in the case of regular graphs is that they do not exist for all choices of degree d and number of vertices n, as is seen from the following proposition.

Proposition 2: d-regular, n vertex graphs exist if and only if dn is even and $d \leq n-1$.

(See [18, Proposition 1], for example, for a proof.) To allow use of Ω-notation in expressing our lower bounds, however, the "time" used by a JAG must be defined for *all* sufficiently large n. To this end, we consider the time used by a JAG on d-regular, n-vertex graphs where dn is odd to be the same as its running time on d-regular, $(n+1)$-vertex graphs. We adopt a similar convention for d-regular symmetrically labeled graphs, which exist if and only if, in addition to the restrictions above, either $d = 0$ or n is even.

It is not difficult to show that st-nonconnectivity requires time $\Omega(m)$ on any of the JAG variants described above, independent of the number of pebbles and states. This result is not surprising, but we will sketch it because of its generality, and also because the proof introduces some ideas we will use subsequently.

Theorem 3: Let n be a multiple of 4, $d < n/2$, and $m = dn/2$. Any JAG, even a nondeterministic one with strong jumping, solving st-nonconnectivity for all symmetrically labeled, d-regular, n-vertex, m-edge graphs requires time $\Omega(m)$ in the worst case.

Proof: With the given constraints on n and d, there is a d-regular, n-vertex, symmetrically labeled graph having its vertices and edges evenly divided between two connected components, one containing s, the other containing t (see [15, Exercise 6.2.1]). Fix a minimal length accepting computation of JAG J on this disconnected graph. Suppose for some $a \in \{0, 1, \ldots, d-1\}$ that pebbles in this computation walk across fewer than $\lfloor m/(2d) \rfloor$ edges labeled a. Then there must be at least one edge labeled a in each component that is not crossed during this computation. These two edges can be cut and rejoined so that the resulting graph is an st-connected graph also accepted by this computation. Hence, J requires at least $m/2$ steps. □

See Theorem 15 for a matching upper bound, which is in fact achieved by a deterministic WAG, even on general graphs.

3. JAGs Have Turing Machine Power

In this section we will show that, although JAGs are structured computational models, they are as "powerful" as Turing machines for the purposes of solving problems about undirected graphs. That is, we will show that any undirected graph problem solvable by a Turing machine is also solvable by a JAG in roughly the same space and time. This holds even on relatively weak variants such as WAGs with one passive and one active pebble. Thus, sufficiently strong lower bounds on JAGs or WAGs will have direct implications for Turing machine complexity.

Since the input conventions for JAGs and Turing machines are quite different, we must specify the correspondence between the two models carefully. For technical reasons, we will focus initially on problems about connected, regular graphs with no distinguished vertices. More general problems, including *st*-connectivity, will be discussed later. Let $\mathcal{G}$ be the set of all edge-labeled, connected, regular graphs, where edges are labeled as described in Section 2. A *graph problem* is simply a subset $\mathcal{H} \subseteq \mathcal{G}$. For example, $\mathcal{H}$ might be the set of (connected, regular, edge-labeled) bipartite graphs, or the set of Hamiltonian graphs. To say that a graph problem $\mathcal{H}$ is solvable by a JAG J has the obvious meaning — J, when started with all its pebbles on an arbitrary vertex of $G \in \mathcal{G}$, accepts if and only if $G \in \mathcal{H}$.

Turing machines, of course, work not on graphs, but rather on encodings of graphs. Thus, to say that $\mathcal{H}$ is solvable by a Turing machine M means that M accepts a "reasonable" encoding of a graph $G \in \mathcal{G}$ if and only if $G \in \mathcal{H}$. To be precise, an encoding is *reasonable* if and only if it is interreducible with the "adjacency matrix" representation by a deterministic Turing machine using $O(\log n)$ space. In the adjacency matrix representation, an n-vertex d-regular graph is represented by a string l of n^2 symbols from the alphabet $\{\star, 0, 1, \ldots, d-1\}$. Let $l(i,j)$ denote the $(n \cdot i + j)^{th}$ symbol of l, $0 \leq i,j \leq n-1$. Then $l(i,j) = l(j,i) = \star$ if and only if vertices i and j are not adjacent, and otherwise $l(i,j)$ is the label on the half edge from vertex i to vertex j. Note that reasonable encodings of graphs (at least implicitly) specify a numbering of the vertices, a feature not present in $\mathcal{G}$. Thus, there may be many different encodings of each graph, corresponding to different vertex numberings. M, of course, must accept all or none of these equivalent encodings.

Consider the following "edge list encoding", which will be used throughout this section. The vertex names are distinct, but not necessarily consecutive, $O(\log n)$ bit integers. An edge is encoded as a triple (i, j, a), where i and j are vertex names and a is the label on the half edge from i to j. The edge list encoding consists of a sequence of such triples, in any order, and possibly with repetitions. It is straightforward to show that this is a reasonable encoding.

The main technical result of this section is that a simple JAG can construct an edge list encoding of its input graph in polynomial time and logarithmic space. This is embodied in Lemma 5 below. One key idea in the proof of Lemma 5 is that a WAG can use a universal traversal sequence (Aleliunas *et al.* [2]) to explore its input. Recall that a universal traversal sequence is guaranteed to visit all *vertices* of a graph. The following simple extension is more useful for our purposes.

A sequence $V \in \{0, 1, \ldots, d-1\}^*$ is said to be a *half edge universal traversal sequence for d-regular, n-vertex graphs* if it has the property that a walk according to V from any start vertex of any d-regular, n-vertex graph G will cross every edge of G at least once in each direction. An analogous definition can be made for nonregular n-vertex graphs of maximum degree d. In this case we define the "walk according to V" so that, at a vertex u of less than maximum degree, the next letter of V selects among u's neighbors evenly. To be precise, when at a vertex u of degree $d(u) = d$, with the next letter of V being $\sigma \in \{0, 1, \ldots, d-1\}$, the walk proceeds to the neighbor v of u having $\lambda_{u,v} = \sigma$, just as in the d-regular case. When $d(u) < d$, the walk *remains* at u if $\sigma \geq \lfloor d/d(u) \rfloor \, d(u)$, and otherwise proceeds to the vertex v having $\lambda_{u,v} = \sigma_i$, where σ_i is the i^{th} smallest label on a half edge leaving u, and $i = \sigma \bmod d(u)$. (A simpler definition of "walk according to V" for nonregular labeled graphs would be to remain at u unless $\sigma = \lambda_{u,v}$ for some v. Under this convention, the bound below would be increased by a factor of $O(n)$.)

Lemma 4: Half edge universal traversal sequences of polynomial length exist for n vertex graphs. In particular, length $O(dn^3 \log n) = O(mn^2 \log n)$ suffices for d-regular graphs, and length $O(m^2 n \log n)$ suffices for all m-edge graphs.

Proof: (Sketch.) The *vertex (half edge) cover time* of a graph G, $C_V(G)$ ($C_E(G)$), is the maximum, over all vertices u, of the expected number of steps required for a random walk starting at u to reach all vertices (cross all half edges, respectively) of G. The *vertex (half edge) hitting time* of G, $H_V(G)$ ($H_E(G)$), is the maximum, over all pairs u, x, of the expected number of steps required for a random walk starting at vertex u to reach vertex x (respectively, to cross half edge x). Clearly hitting time is never greater than cover time, either for vertices or edges. Let $\mathcal{F}$ be a family of edge-labeled graphs, and define $C_V(\mathcal{F})$ to be the maximum cover time of any graph in $\mathcal{F}$, and similarly for $H_V(\mathcal{F})$. A basic result of Aleliunas *et al.* [2] is that any family $\mathcal{F}$ of d-regular graphs has a (vertex) universal traversal sequence of length $O(C_V(\mathcal{F}) \log(n^2|\mathcal{F}|))$. Alon *et al.* [3] and Chandra *et al.* [22] observe that $C_V(\mathcal{F})$ can be replaced by $H_V(\mathcal{F})$ in this expression.

These results extend easily to universal traversal sequences for nonregular graphs of maximum degree d (as defined above) by observing that cover- and hitting times are at most doubled when the random walk is modified so as to remain at a vertex u of degree $d(u)$ with probability $(d - \lfloor d/d(u) \rfloor \, d(u))/d \le 1/2$. Furthermore, for both regular and nonregular graphs, the technique yields an analogous expression bounding the length of half edge universal traversal sequences, using H_E in place of H_V. Zuckerman [50] observes that $H_E(G) \le H_V(G) + 2m$ for all graphs G. Aleliunas *et al.* [2] show that $H_V(G) \le 2m\Delta$, where Δ is the diameter of G (maximum distance between two vertices). It is well known (cf. Lemma 13) that the diameter of d-regular graphs is $O(n/d)$. The Lemma follows, since there are at most n^{dn} labeled d-regular n-vertex graphs, and at most n^{4m} labeled nonregular m-edge, n-vertex graphs. □

We remark that Lemma 4 implies the same bounds for lengths of vertex universal traversal sequences, asymptotically matching the best known upper bounds for both regular (Aleliunas *et al.* [2], Kahn *et al.* [36]) and nonregular graphs.

The main technical result of this section is the following lemma. For the purposes of this lemma, it is convenient to think of the JAG as a "transducer," i.e., as a machine equipped with a one-way, write-only output tape, excluded from the space bound, on which it writes the string encoding the graph given to it as input.

Lemma 5: A deterministic WAG with two pebbles, one of them passive, can output an edge list encoding of its (connected, regular) input graph in time $n^{O(1)}$ and space $O(\log n)$.

Proof: The idea of the proof is for the WAG to use a universal traversal sequence to systematically explore its input G, generating an encoding of the edges it explores as it goes. The key point is to be able to devise a numbering for the vertices, and to determine a vertex's number when needed.

Call the WAG E, and let s be the vertex on which the pebbles of E start. Suppose G is d-regular, with n vertices. Let V be a half edge universal traversal sequence for d-regular, n-vertex graphs. (Cf. Lemma 4.)

Call E's passive pebble p. Initially, E leaves p on s, then determines the shortest prefix U of VV such that $|U| \geq |V|$, and a walk from s according to U ends at s. U has the property that a walk from s according to U returns to s after crossing each edge at least once in each direction. Recall that s is not specially marked in our model. The construction of U allows us to retain s as a landmark without permanently marking it with a pebble.

The vertex number $\#w$ that E assigns to an arbitrary vertex $w \in G$ is the length of the shortest prefix U_w of U such that the walk from s according to U_w ends at w. For instance, $\#s = 0$.

For $1 \leq i \leq |U| + 1$, let v_i be the vertex reached from s by walking according to the length $i-1$ prefix of U. Let a_i be the i^{th} symbol of U. Then, for $1 \leq i \leq |U|$, the half edge crossed during the i^{th} step of the walk according to U from s will be the half edge $\{v_i, v_{i+1}\}$, which has label $\lambda_{v_i,v_{i+1}} = a_i$. During the i^{th} phase of the algorithm, $1 \leq i \leq |U|$, E will determine and write onto its output tape the triple $(\#v_i, \#v_{i+1}, a_i)$ defining this labeled half edge.

Suppose at the start of the i^{th} phase that both E and p are on v_i, and that E has stored in its state the values i and $\#v_i$. (Initially, this holds with E and p on $s = v_1$, $i = 1$, and $\#v_1 = 0$.) During the i^{th} phase, E operates as follows.

1. Carry p across the half edge labeled a_i from v_i, then drop p on the vertex reached (v_{i+1}, by definition).

2. Walk from v_{i+1} according to the last $|U| - i$ symbols of U, thus returning to s.

3. Walk from s according to U until p is encountered. The length of this walk is $\#v_{i+1}$.

4. Output the triple $(\#v_i, \#v_{i+1}, a_i)$ defining this labeled half edge.

Note that, at the completion of this process, E is in the configuration desired for the start of phase $i+1$.

The running time of E is $O(|U|^2) = n^{O(1)}$, and E has $n^{O(1)}$ states. □

On bijectively labeled graphs, it suffices to have only one movable pebble.

Lemma 6: A deterministic WAG with one active pebble, and one arbitrarily placed unmovable pebble can construct a binary string encoding its (connected, regular) bijectively labeled input graph in time $n^{O(1)}$ and space $O(\log n)$.

Proof: The proof is similar to that of Lemma 5. Let U be a half edge universal traversal sequence, and for a vertex w, define $\#w$ to be the length of the shortest prefix of U that walks from w to the fixed pebble. Since the graph is bijectively labeled, these vertex numbers will be unique. (This idea is used by Hoory and Wigderson [32].) For $1 \leq i \leq |U| + 1$, let v_i be the vertex reached from the fixed pebble by walking according to the length $i-1$ prefix of U. Suppose again that the values i and $\#v_i$ are stored in the state, the movable pebble is on v_i, and a_i is the i^{th} symbol of U. Then, for the neighbor v_{i+1} of v_i reached via label a_i, $\#v_{i+1}$ can be determined by walking from

v_{i+1} to the fixed pebble according to U. After writing $(\#v_i, \#v_{i+1}, a_i)$, the movable pebble can be returned to v_{i+1} by walking from the fixed pebble according to the length i prefix of U. Again, the running time of this algorithm is $O(|U|^2) = n^{O(1)}$. $\square$

Theorem 7 below is the main result of this section. It establishes the equivalence between WAGs and "general machines", which include nonuniform Turing machines as a special case. A *general machine* consists of an input $x_1 x_2 \cdots x_n$ and a set of states. The state set may depend on the length n of the input and, in particular, the number of states may be a function of n. Included in each state is the *input index*, which specifies the index of the next input character to be read. In one move, based on its current state q, input index i, and the input symbol x_i, the machine enters a new state q' with a new input index i', as dictated by a transition function that may also depend on n. This transition may be deterministic, nondeterministic, or probabilistic, depending on the type of the general machine. Acceptance is defined as it is for the corresponding types of Turing machines. *Time* is defined as the number of moves, and *space* as $\log_2 Q$, where Q is the number of states.

General machines are almost identical to the "recognition machines" defined by Cobham [23], except that recognition machines require the input to be accessed sequentially, whereas general machines allow completely random access to the input. It is also easy to see that Turing machines are a special case, by including the worktape contents and head positions as part of the state of the general machine.

Theorem 7: Let $\mathcal{H}$ be an undirected graph problem as defined above, and let $S(n) = \Omega(\log n)$. $\mathcal{H}$ is solvable using space $O(S(n))$ and time $n^{O(1)} \cdot T(n)$ by a deterministic (nondeterministic, probabilistic) general machine if and only if it is solvable in space $O(S(n))$ and time $n^{O(1)} \cdot T(n)$ by a deterministic (nondeterministic, probabilistic, respectively) JAG. Moreover, the JAG simulating a general machine requires no jumping and only two pebbles, one of them passive. On bijectively labeled graphs, the WAG requires only one active and one unmovable pebble.

Proof: Let M be a general machine accepting $\mathcal{H}$ in space $S(n)$ and time $T(n)$. By Lemma 5 or 6 above, there is a two pebble deterministic WAG E that can construct a binary string encoding its input graph G. Because of logarithmic space reducibility among reasonable encodings, assume without loss of generality that M operates on the same encoding output by E. We build a WAG W accepting $\mathcal{H}$ by simulating both M and E. Specifically, W maintains M's state as part of its state. If M's input index is i, W simulates E until it generates its i^{th} output bit, and then simulates one step of M (deterministically, nondeterministically, or probabilistically, as appropriate). W continues in this manner until M halts. W's state set must be large enough to encode a state of M and a state of E. This requires $2^{O(S(n))}$ states, or $O(S(n))$ space. Note that it is not necessary to store the string constructed by E; its bits are reconstructed as needed. The simulation by W of each of M's steps requires rerunning the entire computation of E, so W's time bound is $n^{O(1)} \cdot T(n)$.

In the other direction, let J be a JAG that accepts $\mathcal{H}$ using $P(n)$ pebbles $Q(n)$ states, and $T(n)$ time. J is simulated by a general machine M, whose state encodes J's state plus the vertex names on which pebbles currently reside. This requires $Q(n) \cdot n^{P(n)}$ states, or space $P(n)\log_2 n + \log_2 Q(n)$, which is, by definition, J's space bound. M can then simulate a move of J, using its input to determine the vertex name to which a given pebble walks by following a given edge label, which increases the time and number of states by only a polynomial factor. $\square$

Note that a general machine can solve any graph problem in linear space and time (nonuniformly), hence by Theorem 7, a WAG can do so in linear space and polynomial time. Theorems 12 and 15 in Section 4.3 give faster WAG algorithms at this space extreme.

Corollary 8: A JAG solving an undirected graph problem in space $\Omega(\log n)$ can be simulated by a WAG, at the expense of a constant factor loss in space and a polynomial factor loss in time.

Corollary 9: A JAG or WAG solving an undirected graph problem using P pebbles and space $\Omega(\log n)$ can be simulated by one with only two pebbles, at the expense of a polynomial factor loss in time and a constant factor loss in space (more precisely, a factor of $n^{P+O(1)}$ in the number of states).

The polynomial factor loss in time implicit in Corollaries 8 and 9 is $O(U^2(n))$, where $U(n)$ is the length of a half edge universal traversal sequence (Lemma 4). This factor can be improved to $O(U(n))$ by directly using the proof techniques from Lemmas 5 and 6.

As another illustration of Theorem 7, consider the problem of deciding bipartiteness of a connected graph. It is easy to see that a nondeterministic two pebble WAG can recognize *non*bipartite graphs (guess and verify an odd cycle), but not so easy to see a direct way to recognize bipartite graphs. In fact this is also possible, by the following corollary to Theorem 7 and Immerman and Szelepcsényi's Theorem [33, 47].

Corollary 10: Let $\mathcal{H}$ be an undirected graph problem, and let $S(n) = \Omega(\log n)$. If $\mathcal{H}$ is solvable using space $O(S(n))$ by a nondeterministic JAG or WAG J, then so is its complement $\mathcal{G} - \mathcal{H}$.

Proof: Simulate J by a nondeterministic, $S(n)$ space-bounded general machine M. By a straightforward adaptation of Immerman and Szelepcsényi's Theorem [33, 47], there is a nondeterministic, $S(n)$ space-bounded general machine M' that accepts the complement $\mathcal{G} - \mathcal{H}$. Simulate M' by a nondeterministic, $S(n)$ space-bounded WAG. $\square$

We know no substantially simpler method for recognizing bipartite graphs. Implementation of the Immerman/Szelepcsényi method on a JAG seems to require construction of a vertex numbering, which is the key idea in Lemmas 5 and 6.

Algorithmic problems about graphs often have input parameters other than the graph itself. For example, consider the shortest path problem: given a connected undirected graph G, two designated vertices s and t in G, and an integer k, is there a path from s to t of length at most k? The results above are easily extended to encompass such problems by incorporating integers such as k into the WAG's initial state, and marking "designated" vertices or edges with pebbles, or making them "visible" to the WAG as we did for st-connectivity. Thus, for example, the shortest path problem is solvable in (deterministic) logarithmic space by a WAG if and only it is so solvable by a general machine. This problem is of particular interest since it is a problem about undirected graphs that is known to be complete for *non*deterministic logarithmic space (Ladner, personal communication). Hence, it is plausible that complexity results for WAGs will solve a long-standing open problem in Turing machine complexity.

Finally, we mention that the restriction to regular graphs in the above results is only a technicality. Lemmas 5 and 6 are modified easily to accommodate nonregular graphs, since by Lemma 4 there are universal traversal sequences of polynomial length for such graphs. The restriction to connected graphs is only slightly more problematic. Obviously a WAG with only one active pebble cannot explore more of its input graph than the connected component initially holding that pebble. With strong jumping, or with an active pebble in each connected component, or some other mechanism for accessing all components, the results could be extended easily to nonconnected graphs.

4. JAGs with Unmovable Pebbles

A plausible paradigm for an st-connectivity algorithm is to choose and mark a small number of "landmark" vertices in the graph, based perhaps on local properties like proximity to low or high degree vertices or certain small subgraphs, then to explore the graph with these landmarks fixed. This paradigm motivates our study of JAGs with unmovable pebbles.

Depth- and breadth-first search are examples of algorithms where vertices are permanently marked. The undirected st-connectivity algorithms of Broder *et al.* [20], Barnes and Feige [7], and Feige [28] are more complex examples of this paradigm. In outline they operate as follows. First, s and t are marked by pebbles. Then $P-3$ other pebbles are placed on the graph at random. (The random distribution used to place the pebbles is what distinguishes the three algorithms.) These $P-1$ pebbles are not subsequently moved. The one remaining pebble then executes a small number of short random walks from each of the $P-1$ fixed pebbles. At the end of each walk, the movable pebble jumps to one of the fixed pebbles. Connectivity information is inferred from the pebbles encountered during these short walks. For example, if the algorithm has learned that pebbles 1 and 2 are in the same connected component, and similarly for pebbles 3 and 4, and during a walk from pebble 1 the algorithm reaches pebble 4, then it can infer that all four pebbles are in the same component. The authors show that, if s and t are in the same connected component, the algorithm will discover this quickly with high probability. Note that this algorithm can be executed on a model that is essentially a probabilistic JAG, except that the unmovable pebbles are "preplaced" probabilistically without walking to their locations. On a regular graph, the algorithm could be implemented by a probabilistic JAG with strong jumping. On nonregular graphs, the model would have to be extended to allow the dependence of pebble preplacement on vertex degree. In Section 4.2 we will discuss and prove a lower bound for such a model that allows preplacement of pebbles. Prior to that, Section 4.1 gives the lower bound for the simpler basic model, i.e., without preplacement. Section 4.3 shows that this lower bound is tight for the model we consider.

Note that these lower bounds apply to models that are sufficiently rich to admit depth- and breadth-first search, and the algorithms of Broder *et al.*, Barnes and Feige, and Feige. Thus, as corollaries we establish three facts claimed in the introduction — that depth-first search is space-optimal among linear time algorithms, that Feige's algorithm cannot be made both errorless and substantially faster, and that closure under complementation is intrinsically slow (all with respect to this class of models, of course).

4.1. A Lower Bound for Unmovable Pebbles

In this section, we prove an $\Omega(n^2/P)$ lower bound on the time for a nondeterministic P-pebble JAG to solve st-nonconnectivity. We first prove a basic lower bound for regular graphs of degree $d = 3$. Several generalizations are sketched later.

Theorem 11: Let M be any nondeterministic JAG with strong jumping that has 1 active pebble and $P - 1$ unmovable pebbles. If M determines st-nonconnectivity for all 3-regular symmetrically labeled graphs, then M requires time $\Omega(n^2/P)$.

Proof: The proof generalizes the main lower bound technique introduced by Borodin *et al.* [18]. Assume without loss of generality that n is a multiple of 4. (If not, set aside 6 vertices in a 3-regular connected component containing neither s nor t.) We define a family of n vertex graphs, each formed by joining two copies of an $n/2$ vertex graph H by "switching" some combination of edge pairs. We will show that M must frequently walk from one pebble to another via some distant switchable edge.

Many graphs H would work for our purposes; for definiteness, we use the $n/2$ vertex "squirrel cage": two $n/4$ vertex cycles, with each vertex on one cycle joined by an edge, called a "rung," to the corresponding vertex on the other cycle. Fix any numbering of the vertices and any symmetric labeling of the edges of H. Take as the set of "switchable" edges any $r = n/4 - 1$ of the rungs. As in Borodin *et al.* [18], for each $x \in \{0,1\}^r$ the graph G_x is formed from two copies H^0 and H^1 of H by "switching" the edges corresponding to the 1's in x. That is, if $\{u^0, v^0\}$ is the i^{th} switchable edge in H^0 and $\{u^1, v^1\}$ is the corresponding edge in H^1, then G_x has the pair of edges $\{u^b, v^{b\oplus x_i}\}, b \in \{0,1\}$, with labeling $\lambda_{u^b,v^{b\oplus x_i}} = \lambda_{u,v} = \lambda_{v,u} = \lambda_{v^{b\oplus x_i},u^b}$, where $\oplus$ denotes the EXCLUSIVE OR operation. Choose s to be any vertex in H^0 and t any vertex in H^1. Let $\mathcal{G} = \{G_x \mid x \in \{0,1\}^r\}$. Notice that the only graph in $\mathcal{G}$ with no path from s to t is G_{0^r}, and that all graphs in $\mathcal{G}$ are symmetrically labeled.

The key property of the family $\mathcal{G}$ of graphs is that a walk on G_x is identical to such a walk on G_{0^r}, except that the walk switches from one copy of H to the other when a switched edge is crossed. After any sequence of edge crossings starting from a vertex v, M's active pebble will be in the same copy of H as v exactly when the net parity with respect to x of all edge crossings is even, where the *parity with respect to* x of an individual edge e is defined to be x_i if e is the i^{th} switchable edge, for any $1 \leq i \leq r$, and 0 for all unswitchable edges.

Intuitively, M gains information about connectivity only by *walking* to a pebble; nothing is learned (directly) about the existence or nonexistence of a path from u to v by jumping from u to v. We exploit this fact, together with the fact that pebbles on average are far apart, to argue that M must execute many walking steps.

Note that s and t are *not* connected in G_{0^r}, hence M must have at least one accepting computation on G_{0^r}. Fix one such computation γ of minimal length. Assume that two extra unmovable pebbles are placed on the distinguished vertices s and t. Now in G_{0^r} "mark" both copies of each vertex that received an unmovable pebble during the computation γ. Break γ into sequences of walk moves that (1) begin with a walk move from a vertex that either is marked or was the target of a jump in the immediately preceding step, and (2) end with the next walk move into a vertex that either is marked, is the source of a jump move in the immediately following step, or is the last move of γ. Discard any such sequence that does not end at a marked vertex. Suppose there are w sequences remaining. Each of these sequences naturally corresponds to a connected sequence of edges in G_{0^r}. Notice that if, for some x, one of the w sequences is of odd parity with respect to x, then the computations of M on G_{0^r} and on G_x may diverge at the end of this sequence, since a pebble encountered on one may not be encountered on the other. This cannot occur if all sequences have even parity with respect to x:

Claim: For every $x \in \{0,1\}^r$, if each of these w edge sequences is of even parity with respect to x, then γ is also an accepting computation for G_x.

To see this, we show by an induction on i, that after making the moves dictated by γ up to the end of the i^{th} sequence (including the discarded sequences), the configurations of M on G_{0^r} and on G_x are identical, with the exception that the movable pebble will be on opposite copies of a vertex if the net parity with respect to x of the i^{th} sequence is odd. (This can happen only if this is a discarded sequence.) Basically, this is true since all the "interesting" events in the computation γ, i.e., dropping or encountering pebbles, occur at marked vertices, and we've taken care that all walks between these interesting points are of even parity in G_x just as they were in G_{0^r}. The base case ($i = 0$) is vacuous. For the induction step, first note that the configurations at the start of the i^{th} sequence are the same on both graphs, since if they differed at the end of the $(i-1)^{st}$, then all intervening steps were jumps. All steps within the i^{th} sequence are walk steps into unmarked vertices, hence no pebbles are encountered during those steps in either G_{0^r} or G_x. Since both graphs are 3-regular, all unpebbled vertices "look alike", so the i^{th} sequence of walk moves of γ in G_{0^r} is also a legal sequence of moves in G_x, and carries the movable pebble to the same place in both graphs, up to the parity of the sequence with respect to x. This completes the proof of the claim.

As noted earlier, G_x is connected for all $x \neq 0^r$, hence must not be accepted by M. Thus it must be that there is no $x \neq 0^r$ for which the w sequences all have even parity. Equivalently, it must be that the corresponding homogeneous system of w linear equations in r unknowns over $GF(2)$ has no nonzero solution.

Let S be the set of r switchable edges. For each $e \in S$, let $\text{dist}(e)$ be the distance from e to the closest marked vertex, where the distance from an edge to a vertex is defined to be the length of a shortest path containing both. Let m be the maximum integer such that some switchable edge e has $\text{dist}(e) = m$. For any nonnegative integer d, let $S_d = \{e \in S \mid \text{dist}(e) \geq d\}$, and let r_d be the number of switchable edges e with $\text{dist}(e) = d$, so that $r_d = |S_d| - |S_{d+1}|$.

Now it must be the case that, for all $d \leq m$, at least $|S_d|$ of the w walks each have length at least d. If this were not the case, then the edges in S_d would appear collectively on fewer than $|S_d|$ walks or, equivalently, the variables corresponding to these edges would occur in fewer than $|S_d|$ of the homogeneous equations. Set the variables corresponding to the other $r - |S_d|$ switchable edges to 0, and these $|S_d|$ to some nonzero solution, which must exist in a homogeneous system with fewer equations than unknowns (Herstein [31, Corollary to Theorem 4.3.3]). Since such a nonzero solution cannot occur, we have a contradiction.

Thus, at least r_m of the w walks each have length at least m, an additional r_{m-1} each have length at least $m-1$, etc. In other words, M makes at least

$$\sum_{d=1}^{m} d \cdot r_d = \sum_{e \in S} \mathrm{dist}(e)$$

moves. This last sum is minimized when the $O(P)$ marks are equidistantly distributed around the cycle, in which case the sum is $\Omega(rn/P) = \Omega(n^2/P)$. □

Using Hall's Theorem [29], one can in fact prove somewhat more about the w walks: each switchable edge in S can be assigned a unique walk that contains it.

Next, we will sketch several promised generalizations to the theorem. First, to extend the result to d-regular graphs, $d \geq 3$, we generalize the squirrel cage graph H. Note that K_{d-1}, the $d-1$ vertex complete graph, is $d-2$ regular. Form the new d-regular, $n/2$ vertex graph H from $(d-1)$ cycles of length $n/(2(d-1))$ by joining corresponding groups of $(d-1)$ vertices as K_{d-1}. (An extra gadget is needed if $2(d-1)$ does not divide n.) The rest of the argument is essentially as before, except that there are more switchable edges (all but a spanning tree of H, hence $\Theta(dn)$ of them), but on average they are closer to marked vertices ($\Omega(n/(dP))$ average distance). The result is still an $\Omega(n^2/P)$ lower bound, independent of d. To provide some intuition of why the bound does not increase with d, note that any connected d-regular graph has diameter $O(n/d)$, a corollary of Lemma 13 below. This idea is exploited in Theorem 12 to obtain a matching upper bound.

A better bound is possible for nonregular graphs. For n vertex graphs of maximum degree d, one can prove an $\Omega(dn^2/P)$ lower bound, provided $n/(4d) \geq 2P$. Again, the key point is to choose H appropriately. In this case it suffices to take H to be an $n/4$ vertex cycle, attached at evenly spaced intervals to $n/(4d)$ copies of K_d. Most of the $\Theta(dn)$ edges are switchable, and their average distance from any placement of P pebbles is $\Omega(n/P)$. In Section 4.3 we prove matching upper bounds for both the regular and nonregular cases, demonstrating that this disparity in bounds is inherent in the problem.

The remaining generalization promised above is to the case where the automaton can move the "unmovable" pebbles a limited number of times. (A detail about the algorithms of Broder *et al.*, Barnes and Feige, and Feige that we oversimplified above is that they rerandomize the placement of the $P-3$ landmark pebbles $O(\log n)$ times.) Suppose M is a P-pebble JAG of this more general form. Suppose pebbles are placed on at most P' vertices during M's computation. Then a straightforward adaptation of the proof of Theorem 11 shows that M requires time $\Omega(n^2/P')$. Note that, as long as the number of pebble placements is sublinear, the time must be superlinear. However, any graph in the family $\mathcal{G}$ built from squirrel cage graphs as above can be traversed in linear time by a deterministic automaton with 2 pebbles, one of them passive, even without jumping, provided the passive pebble can be moved freely. Thus, stronger proof techniques are necessary for freely moving pebbles; see our companion paper [9] for one such technique.

4.2. Preplacement of Unmovable Pebbles

As we have noted earlier, the JAG is a powerful yet restricted model. It is conceivable that there is certain useful information about graphs that is intuitively "easy to compute," yet hard for JAGs to compute. That is, there might be certain information about an input graph G that (1) could be collected easily by a more flexible computational device such as a logarithmic space RAM, that (2) would greatly facilitate a JAG's determination of the st-nonconnectivity of G, yet (3) is difficult or impossible for a JAG to collect. If this were the case, it might "explain" (and trivialize) the strong lower bound given in the previous section.

The algorithms of Broder *et al.*, Barnes and Feige, and Feige again furnish a motivating example. In all three algorithms the initial (random) pebble placement is dependent on vertex degree. On nonregular graphs, a JAG cannot duplicate this behavior without visiting all vertices, a slow or even impossible process for, say, a probabilistic JAG without strong jumping. Yet this is an easy process for a RAM, and a crucial one for the efficiency of their algorithms. (Note that the rest of their algorithms *can* be performed efficiently by a JAG.) Generalizing this slightly, it might be useful to know how many neighbors each vertex has at distance two. Although this information is easily computed by a RAM, as far as we know it is not easily computable by a JAG with one active pebble and a limited number of unmovable pebbles, even a nondeterministic one with strong jumping.

Does our lower bound rest on this or similar deficiencies of the JAG model? In this section we give evidence that it does not. We generalize the model to allow precomputation on the input and preplacement of (unmovable) pebbles, and show that a similar lower bound holds. Of course, such precomputation must be restricted so as to preclude solving st-connectivity itself. Therefore, the unmovable pebbles are placed based on complete knowledge of the local, but not global, structure of the graph as described below.

Let $N_\rho(G)$ denote a list $G_1, G_2, \ldots G_n$ of edge labeled graphs, each with a distinguished vertex, such that G_i is isomorphic to the radius ρ neighborhood of vertex i in G, and the isomorphism maps G_i's distinguished vertex to vertex i. For instance for a triangle free graph, and ignoring edge labels, $N_1(G)$ is equivalent to an ordered list of the degrees of G's vertices. Then an automaton *with P' unmovable pebbles placed by ρ-precomputation* is a pair (f, M), where M is one of the JAG variants as described above, and f is an arbitrary function mapping $N_\rho(G)$ to $U \in \{1, 2, \ldots, n\}^{P'}$. Given G, the P' unmovable pebbles are placed on the sequence of vertices $f(N_\rho(G))$, and then M is run on the resulting pebbled graph. The definition can be further generalized to allow f to select M's initial state. Additionally, it can be generalized in a straightforward way to probabilistic or nondeterministic precomputation by letting f be a relation, and selecting a value from its range probabilistically or nondeterministically. For instance, the algorithms of Broder *et al.*, Barnes and Feige, and Feige can be executed by a probabilistic JAG with probabilistic 1-precomputation.

The proof of Theorem 11 immediately extends to show an $\Omega(n^2/P)$ lower bound on nondeterministic JAGs with nondeterministic 1-precomputation. The only changes needed in the proof are to note that the initial pebble placement and state $f(N_1(G_{0^r}))$ are considered to be part of the fixed accepting computation γ, and to note that all graphs in $\mathcal{G}$ are 3-regular, symmetrically labeled, and triangle-free, hence $N_1(G_{0^r}) = N_1(G_x)$ for all $x \in \{0,1\}^r$, and so this initial configuration is also legal in G_x.

As a concrete example of the potential utility of precomputation, we note that the squirrel cage family $\mathcal{G}$ defined above can be traversed quickly, provided the unmovable pebbles can be placed based on vertex neighborhoods of radius two, generalizing the use of vertex degree. Specifically, in G_x, a vertex v will have 4, 5, or 6 distinct neighbors at distance two depending on whether the "rung" of the squirrel cage incident to v is of the same parity as both, one, or neither, respectively, of the two nearest nonincident rungs. Thus, 2-precomputation *alone* suffices to distinguish the disconnected graph G_{0^r} (every vertex has 4 neighbors at distance two) from all the connected members of $\mathcal{G}$ (some vertex has more than 4 neighbors). Furthermore, by placing one unmovable pebble on any vertex with more than 4 neighbors at distance two, a WAG with no additional pebbles can traverse the entire graph in linear time.

However, we can show that radius two, or indeed any constant radius, does not help in general. That is, we can further generalize the proof of Theorem 11 to use families of graphs in which switched edges do not alter the local structure within any fixed radius ρ. This is done by choosing a d-regular bipartite graph R whose girth (minimum cycle length) is at least $2\rho + 2$ and whose size $|R|$ is $d^{O(\rho)}$ (Bollobás [14, Chapter 3]), and then constructing the half-size graph H by connecting $c = \lfloor n/(2|R|) \rfloor$ copies of R in a cycle. One way to do this is to choose a fixed edge $\{u, v\}$ in R, remove this edge from each copy of R, then insert an edge from u in the i^{th} copy of R to v in copy $(i + 1) \bmod c$, $0 \le i < c$. Note that, for every cycle in G_x, there is a corresponding cycle in G_{0^r} that is no longer, so all graphs in $\mathcal{G}$ have girth at least $2\rho + 2$. Furthermore, note by Hall's Theorem [29] that R can be symmetrically labeled since it is regular and bipartite, hence so can G_{0^r}. The key new idea in the proof is that the list $N_\rho(G)$ of radius ρ neighborhoods of any symmetrically labeled, d-regular, girth $2\rho + 2$ graph G will simply consist of n identical symmetrically labeled, degree d, complete trees of height ρ. Thus, ρ-precomputation cannot distinguish between G_{0^r} and G_x. The remainder of the proof is essentially unchanged. Thus, nondeterministic JAGs with nondeterministic ρ-precomputation require time $n^2/(d^{O(\rho)}P)$ to solve st-nonconnectivity for d-regular graphs.

We remark in closing this section that ρ-precomputation seems to be orthogonal to pebble placement by the JAG itself. For instance, as noted above, deterministic 2-precomputation may be helpful even to a nondeterministic JAG with strong jumping on as simple a family as the basic squirrel cage family. On the other hand, there are cases where even a weak model such as a deterministic WAG can place pebbles more effectively than can be done by deterministic precomputation. Specifically, we again consider the simple 3-regular squirrel cage family, but enlarged to include all $n!$ permutations of vertex labels for each G_x, $x \in \{0,1\}^r$. Suppose all unmovable pebbles are placed by deterministic 1-precomputation. Then an $\Omega(n^2)$ lower bound applies for $P \le n - \Omega(n)$, since all pebbled vertices might be concentrated on one part of the squirrel cage pair. On the other hand, a deterministic WAG (knowing the edge labeling) can easily walk one of the cycles, dropping its $P - 1$ unmovable pebbles at evenly spaced positions around the cycle. It is then a simple matter to test the switchable edges one after the other from the nearest pebble, hence solving st-connectivity in time $O(n^2/P)$.

4.3. An Upper Bound for Unmovable Pebbles

A natural question to ask is whether the lower bounds given in Section 4.1 can be improved. Recall that Theorem 11 shows time $\Omega(mn/P)$ ($\Omega(n^2/P)$ for regular graphs) is required by JAGs with unmovable pebbles and strong jumping, even with an unbounded number of states. We will close this section by showing that this bound cannot be improved: on the model to which the lower bounds apply, exploiting an unbounded number of states we give matching upper bounds on time for a given number of pebbles, even without jumping. More strongly, we show that *any* graph problem, as defined in Section 3, can be solved within the same bounds.

Theorem 12: Let $\mathcal{G}$ be the set of all bijectively labeled graphs (all bijectively labeled regular graphs). For any $P \geq 2$, the following sets can be recognized by a nondeterministic WAG with one active pebble, $P-1$ unmovable pebbles, an unbounded number of states, and time $O((mn/P)+m)$ ($O((n^2/P)+m)$ in the case of regular graphs):

1. the set of st-nonconnected graphs in $\mathcal{G}$, or
2. any set $\mathcal{H}$ of connected graphs in $\mathcal{G}$.

The main import of this result is to show the limits of the proof technique used in Theorem 11. For example, we do not believe that st-nonconnectivity can be solved by a nondeterministic JAG in time $O(mn)$ and space $O(\log n)$ simultaneously. The fastest known logarithmic space nondeterministic JAG for st-nonconnectivity is much slower than this. Indeed, no better method is known than to use a universal traversal sequence, i.e., a deterministic one pebble WAG, which may require time $\Omega(m^2 n \log n)$ for nonregular graphs (Lemma 4). However, Theorem 12 shows that to obtain a lower bound greater than that of Theorem 11 we must somehow exploit a bound on the number of states, as well as the number of pebbles. (It might also be possible to exploit nonbijective labelings but, in light of Lemma 1 and the remarks following the proof of the theorem, this issue is a technicality of the model that is not of fundamental importance to the computational complexity of st-connectivity.)

The following facts are needed in the proof of Theorem 12.

Lemma 13: Let G be a connected d-regular graph, u and v be any two vertices in G, and $\text{dist}(u,v) = l$ be the length of a shortest path between them. Then there are at least $(d+1)\lfloor (l+2)/3 \rfloor$ vertices in G within distance l of u.

Proof: Let $\Gamma(x) = \{ y \mid \text{dist}(x,y) \leq 1 \}$. Fix a shortest path $u = u_0, u_1, \ldots, u_l = v$ from u to v. Then $\Gamma(u_0), \Gamma(u_3), \ldots, \Gamma(u_{3\lfloor (l-1)/3 \rfloor})$ are pairwise disjoint, for otherwise there would be a shorter path from u to v. Furthermore, these sets are all of size $(d+1)$, and all are within distance l of u. □

Corollary 14: Let G be a connected d-regular graph, and s a vertex in G. For any positive integer $P \leq n/d$, there exists a set S with $s \in S$ and $|S| \leq P$ such that every vertex of G is within distance $l = 2\lceil 1 + 3n/((d+1)P) \rceil = O(n/(dP))$ of some member of S.

Proof: Construct $S = \{s_0, s_1, \ldots\}$, where $s_0 = s$ and, for $i \geq 1$, s_i is chosen to be any vertex at distance greater than l from $\{s_0, \ldots, s_{i-1}\}$. The neighborhoods of radius $l/2$ around the s_i's are pairwise disjoint. Furthermore, by Lemma 13, each of these neighborhoods will be of size at least $(d+1)\lfloor (l/2+2)/3 \rfloor \geq n/P$. Hence at most P members of S can be chosen before no vertices of G remain at distance greater than l. □

The analogous results for nonregular graphs are that at least $(l+1)$ vertices are within distance l of u, hence P vertices can be chosen so that every vertex is within distance $2n/P$ of a chosen vertex. The proofs are similar, but easier.

Finally, we prove the theorem.

Proof (of Theorem 12): The approach is to nondeterministically guess the graph, then verify the guess. First we prove part 1: we describe a nondeterministic WAG M accepting st-nonconnected graphs.

Let G be the input graph. It suffices to verify that the connected component C of G containing s does not contain t. Let $l = 2n/(P-1)$, or $l = 2\lceil 1 + 3n/((d+1)(P-1))\rceil$ in the case of regular graphs. By Corollary 14, for any n-vertex graph G and designated vertex s, there is a set of $P-1$ vertices including s such that every vertex of C is within distance l of a member of this set. Leave one unmovable pebble on s (the initial location of the active pebble), and place the other $P-2$ unmovable pebbles on arbitrary, distinct vertices selected nondeterministically during a walk γ of length at most $2(n-1)$ from s back to s. (This walk is long enough to traverse a spanning tree of C, hence any vertex may be pebbled.)

M proceeds by guessing and recording in its state an $n' < n$ vertex, connected, bijectively labeled graph B with $P-1$ distinct vertices marked by numbered pebbles. The remainder of M's computation is deterministic. In outline, M constructs a mapping ϕ from B to C, then verifies that ϕ is a surjective homomorphism. That is, ϕ is a surjection preserving pebble placement, vertex degree, adjacency, and edge labeling. Thus, for all vertices u in B, (1) there is a pebble p on $\phi(u)$ in C if pebble p is on u in B, (2) degree(u) = degree$(\phi(u))$, and (3) for all edges $\{u,v\}$ in B, if $\lambda_{u,v} = a$ then $\lambda_{\phi(u),\phi(v)} = a$. (It might seem more natural to guess an *iso*morphic graph B, and it would not be difficult to modify M to do this, but a homomorphism suffices and is easier to verify.) To complete the algorithm, M visits $\phi(v)$ in C for all $v \in B$, accepting if and only if none is the specially marked vertex t. (Recall that M can sense when it has a pebble on t.)

We now show how to construct and verify the homomorphism ϕ. A key property of a bijectively labeled graph, used earlier in Lemma 6, is that for any sequence σ of edge labels, and any vertices u, u' and v, if walks following σ from both u and u' end at v, then $u = u'$. Otherwise, the graph is nonbijectively labeled at the vertex where the two paths last converge. This property is central to constructing and verifying the homomorphism. In particular, recall that a JAG has no access to vertex numbers of the input graph. Instead, we will identify vertices by paths to or from pebbles.

M now runs a breadth-first search of B, with the queue initially containing all the pebbled vertices. The result is a spanning forest of B with B's pebbles as the roots. Reject if any tree has height greater than l. Otherwise, for all vertices u in B, let $\rho(u)$ be the number of the pebble marking the root of the tree containing u, let $\sigma(u)$ be the sequence of edge labels on the unique path of tree edges from $\rho(u)$ to u, and let $\sigma^{-1}(u)$ be the sequence of labels in the reverse direction, i.e., from u to $\rho(u)$. For all vertices u in B, define $\phi(u)$ to be the vertex reached in C by walking from the vertex marked by pebble $\rho(u)$ according to the sequence $\sigma(u)$.

M now performs the following test.

For all edges $\{u,v\}$ in B, say with labels $\lambda_{u,v} = a$ and $\lambda_{v,u} = b$, M verifies that in C the walk $\sigma(u)a\sigma^{-1}(v)$ ends at $\rho(v)$ when started from $\rho(u)$, and that $\sigma(v)b\sigma^{-1}(u)$ returns to $\rho(u)$ from $\rho(v)$. During this process, at the first visit to $\phi(u)$ for each u in B, M also verifies that degree(u) = degree$(\phi(u))$ (with the same set of labels).

We now show that ϕ is a surjective homomorphism if and only if this test succeeds. First, suppose ϕ is a surjective homomorphism. For any vertices x and y in B, if a walk from x according to α ends at y, then a walk from $\phi(x)$ in C according to α must end at $\phi(y)$. This is shown easily by induction on the length of α, using the fact that $\lambda_{u,v} = \lambda_{\phi(u),\phi(v)}$ for all edges $\{u, v\}$. By construction, for any edge $\{u, v\}$ in B, the walk in B from $\rho(u)$ according to $\sigma(u)\lambda_{u,v}\sigma^{-1}(v)$ must end at $\rho(v)$. Furthermore, by construction, if vertex w in B holds a pebble, then $\phi(w)$ holds the same pebble in C. Consequently, each of M's "walk" tests will succeed. By the assumption that ϕ is a homomorphism, each of M's degree tests will also succeed, and so ϕ passes the test.

Conversely, suppose the test succeeds. We argue that ϕ is a surjective homomorphism. Note that by construction a walk from $\rho(v)$ according to $\sigma(v)$ ends at $\phi(v)$, for all v. We claim first that the reverse also holds: a walk from $\phi(v)$ according to $\sigma^{-1}(v)$ ends at $\rho(v)$, for all v. If v has a pebble, this is trivial. Otherwise v is the child of some u in the spanning forest of B. Then $\rho(v) = \rho(u)$ and $\sigma(v) = \sigma(u)a$ for some a. Since the test succeeds, $\sigma(u)a\sigma^{-1}(v)$ goes from $\rho(u) = \rho(v)$ to $\rho(v)$. But the first part $\sigma(u)a$ goes from $\rho(v)$ to $\phi(v)$, so the last part $\sigma^{-1}(v)$ must go from $\phi(v)$ to $\rho(v)$, establishing the claim. Note that as a consequence, if a walk in C from a vertex w according to $\sigma^{-1}(v)$ ends at $\rho(v)$, then $w = \phi(v)$, since C is bijectively labeled.

The following properties of ϕ are now easily established.

1. For all pebbled vertices u in B, $\phi(u)$ holds the same pebble. This holds by construction.

2. For all $u \in B$, degree(u) = degree$(\phi(u))$. This holds since M explicitly tests for this condition, and by assumption the test succeeds.

3. For all adjacent vertices $u, v \in B$, $\phi(u)$ and $\phi(v)$ are adjacent, with $\lambda_{\phi(u),\phi(v)} = \lambda_{u,v}$ (and $\lambda_{\phi(v),\phi(u)} = \lambda_{v,u}$). This holds since $\sigma(u)\lambda_{u,v}\sigma^{-1}(v)$ walks from $\rho(u)$ to $\rho(v)$, by construction the vertex reached by $\sigma(u)$ is $\phi(u)$, and by the remark above the vertex from which $\sigma^{-1}(v)$ reaches $\rho(v)$ is $\phi(v)$.

4. ϕ is surjective. If this did not hold, there would be a vertex in C *not* in the range of ϕ that is adjacent to a vertex $\phi(u)$ that *is* in the range of ϕ. However, this is impossible, since by property 2, degree(u) = degree$(\phi(u))$, and by property 3, $\phi(u)$ has degree(u) neighbors that *are* in the range of ϕ.

Thus, ϕ is a surjective homomorphism, as claimed.

If M accepts, then there is an accepting computation in which B is isomorphic to C, hence has at most m edges. In this computation, the algorithm makes $O(m)$ walks, each of length $O(l)$, hence the total running time is $O(ml)$, as desired. Note that M can move between the pebbles in C by walking γ, the tour used initially to drop the pebbles, which adds only $O(n)$ to the time.

The proof of part 2 of the theorem is similar. The main difference is that the graph B guessed by M will have n vertices, rather than $n' < n$. M then verifies that this graph is isomorphic to the input graph G, accepting (nonuniformly) if and only if it is in $\mathcal{H}$. Note that the homomorphism test given above suffices to verify that B is isomorphic to G, since they have the same number of vertices. □

As in Section 3, these results can be generalized to graph problems with other input parameters, and/or to other problems about unconnected graphs, given an appropriate mechanism for accessing all connected components.

The restriction of Theorem 12 to bijectively labeled graphs can be relaxed at the expense of adding one passive pebble, as follows. The constructions of $\phi(u), \sigma^{-1}(u), \sigma(u)$, and $\rho(u)$ are as before. With a nonbijectively labeled graph it remains true that a walk from $\phi(u)$ according to $\sigma^{-1}(u)$ will end at $\rho(u)$, but it is no longer true that $\phi(u)$ is the only vertex with this property. To verify that the active pebble is on vertex $\phi(u)$, we instead leave the passive pebble there, then verify that $\sigma^{-1}(u)$ walks to $\rho(u)$, from which $\sigma(u)$ returns to the passive pebble. The remainder of the algorithm is unchanged.

As a final observation, the following theorem shows that, at the extreme where $P = n$, the WAG of Theorem 12 can be made deterministic.

Theorem 15: The set of st-nonconnected graphs, and arbitrary sets of connected graphs (non-regular, under general labelings) can be recognized in time $O(m)$ by a deterministic WAG with one active pebble and n unmovable pebbles.

Proof: Rather than guessing the input graph, as in Theorem 12, the WAG simply does a systematic traversal of it, akin to a depth-first search, placing a pebble on each vertex. With jumping, or with symmetric edge labels, depth-first search itself would be easy to implement, but lacking both it seems difficult to quickly return after crossing a "back edge" whose reverse label is unknown. We avoid this problem with the following algorithm, which is also akin to an algorithm for finding Euler tours.

M places a distinctly labeled pebble on each vertex it visits, thus effectively numbering the vertices. M records in its state the source, destination, and label of each half edge it crosses. It will eventually cross each half edge, so at termination it will have in its state a complete description of the graph. Connectivity or other properties of the graph can then be determined directly (nonuniformly).

M starts at s, initializing a stack in its state to contain s. At a general step, when at a vertex u with u on top of the stack, if there is a previously uncrossed half edge leaving u, say (u, v), then M crosses this edge, pushing v onto the stack. (M pebbles v if it does not already hold a pebble.) If there are no previously uncrossed half edges leaving u, then M backtracks by popping u from the stack, and returning to v, where v is the new top of stack. By assumption M has previously crossed the (u, v) half edge, and so knows its label. In either case, the process is repeated at v. M terminates when the stack is emptied.

It is easy to see that every visited vertex is pushed onto the stack, and none is removed from the stack until all its outgoing half edges have been traversed. Thus, M will visit all vertices reachable from s. M's running time will be exactly $4m$, since exactly two moves can be charged to each half edge (u, v) — one for the first move by M across that half edge, when v is pushed (on top of u), and the second for the move across (v, u) when that instance of v (there may be several instances) is popped from the stack. $\square$

As noted above, with jumping it would be easy to implement depth-first search directly in $O(m)$ time using $O(n)$ pebbles, and space $O(n \log n)$ in total. The algorithm presented in the proof of Theorem 15 also uses $O(n)$ pebbles, but uses more space, namely $\Theta(m \log n)$ in total, since it constructs a representation of the entire graph in its state. It is not known whether the result can be strengthened to match the bounds attained by depth-first search while retaining the weaker model assumed in Theorem 15.

5. Lower Bounds for the Cycle

5.1. A Lower Bound on the Number of States

In this section we show that deterministic nonjumping automata with a constant number Q of states, one active pebble, and a constant number P of passive pebbles are too weak for studying lower bounds on time. In fact, unless $PQ = \Omega(n)$ such automata cannot even traverse all n-vertex cycles, no matter how much time they are allowed.

Lemma 16: Let $\alpha \in \{0,1\}^*$. Consider the chain C_α of length $2|\alpha|$ with left endpoint L, right endpoint R, and midpoint M, and edge labels so that α is the labeling from L to M and also from R to M. Then starting at any vertex v on C_α that is an even distance from L and traversing according to α terminates at M.

Proof: Consider three pebbles traversing simultaneously according to α, beginning at L, v, and R, respectively. A straightforward induction shows that the pebble that began at v is always an even distance from the other two and between them. Since the ones that started at L and R both end at M, so does the third. $\square$

Theorem 17: Any WAG W that traverses every labeled n-cycle using Q states, one active pebble, and P passive pebbles satisfies $(P+4)Q \geq n$.

Proof: Assume to the contrary that $(P+4)Q < n$. Consider the action of W's active pebble if it never encounters a passive pebble it previously dropped: it traverses according to the sequence $t = \alpha_0\alpha_1 \cdots \alpha_P \in \{0,1\}^*$, where α_i is its traversal after dropping i but before dropping $i+1$ pebbles. If each $|\alpha_i| \leq Q$, then $|t| \leq (P+1)Q < n-1$, so that W does not traverse any cycle having t as the prefix of the clockwise labeling beginning at the start vertex. Thus let i be the least integer such that $|\alpha_i| > Q$. Then W repeats some state during this interval, and $\alpha_i = \rho\beta\beta\beta\cdots$ is infinite, with $|\rho| + |\beta| \leq Q$.

Let $\alpha = \beta\beta$ and $t' = \alpha_0\alpha_1 \cdots \alpha_{i-1}\rho$. Consider the cycle in which t' is the clockwise labeling from the start vertex to a vertex L, followed by an embedding of the chain C_α of Lemma 16 from L clockwise to R. Notice that $|t'| + |C_\alpha| \leq (P+4)Q < n$, so that this labeling can be embedded on a cycle of length n. Now a traversal according to $t'\alpha$ causes the active pebble to move unidirectionally to the midpoint M of C_α, so that no pebble dropped is reencountered. By Lemma 16, each further traversal according to α returns to M, so that the pebbles previously dropped cannot be reencountered, and R is never reached. $\square$

In contrast, it is easy to see that there is a nonjumping automaton that traverses every labeled n-cycle using a constant number of states and only 2 *active* pebbles, and in addition requires only $O(n)$ time. The idea is to maintain the invariant that the leading and trailing pebbles are on adjacent vertices, and the automaton knows the label from the trailing pebble to the leading pebble. Now after moving the leading pebble along label 0 it is a simple matter to advance both pebbles one vertex while maintaining the invariant. A similar construction works with only one passive pebble, if the automaton can jump.

Cook and Rackoff [24, Theorem 4.14] present a family of 3-regular graphs that cannot be traversed using a constant number of states and pebbles, even if jumping is allowed and the edge labels are disclosed. The price paid to capture this strengthened model is a bound that is quantitatively weaker than that of Theorem 17. For instance, they do not rule out the combination $Q = O(1)$ and $P = O(\log\log n)$.

5.2. The Form of Universal Traversal Sequences

As another byproduct of Lemma 16, there is an interesting corollary concerning universal traversal sequences for the cycle. It is not clear *a priori* that a sequence such as $(00010)^{n^2}$ could not be universal for all cycles. The following corollary of Lemma 16 shows that this is impossible.

Corollary 18: For any $\alpha \in \{0,1\}^+$ and any integers n and k, if $|\alpha| < n/2$ then α^k is not a universal traversal sequence for all labeled n-cycles.

Proof: Since $|\alpha| < n/2$, the chain C_α of Lemma 16 can be embedded in a cycle of length n. Consider a traversal according to α starting at M. If $|\alpha|$, the distance from M to L, is even then, according to Lemma 16, the traversal ends at M. If $|\alpha|$ is odd then the traversal ends at a vertex an even distance from L, so that a second traversal according to α returns to M. In either case a traversal according to $\alpha\alpha$ starting at M returns to M after visiting at most $|\alpha| + 1 < n$ distinct vertices. Therefore α^k starting at M never visits more vertices. $\square$

Using similar techniques, Theorem 19 proves that the previous result in fact holds for any even length α such that α is not a universal traversal sequence for all labeled $(n/2)$-cycles. For instance, it holds for any α whose length is even and $O(n^{1.43})$ (Buss and Tompa [21]).

Theorem 19: For any $\alpha \in \{0,1\}^*$ of even length, any even integer n, and any integer k, if α is not a universal traversal sequence for all labeled $(n/2)$-cycles, then α^k is not a universal traversal sequence for all labeled n-cycles.

Proof: Since α is not a universal traversal sequence for all labeled $(n/2)$-cycles, there is a labeled chain C of $n/2 - 1$ vertices with a vertex S such that starting at S and traversing according to α never leaves C and ends at some vertex T. Construct a cycle of length n as follows (see Figure 1): take a copy of C in which T is clockwise from S, followed by a new vertex M, followed by a copy C' of C in which the copy T' of T is counterclockwise from the copy S' of S, followed by a new vertex X.

Now start at any vertex s on the arc between S and S' containing M, where s is an even distance from S, and traverse according to α. This must terminate at a vertex t on the arc between T and T' containing M, where t is also an even distance from S, without ever reaching X. The reason t is between T and T' is that the walk from s to t is trapped between the walks from S to T and from S' to T'. The reason t is an even distance from S is because s is, and because $|\alpha|$ is even.

Therefore, starting at S and traversing according to α^k will never reach X, for any k. $\square$

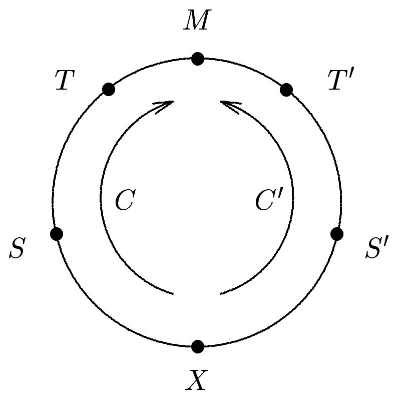

Figure 1: A Cycle Constructed from a Chain and its Reversal

6. Conclusion

This paper has investigated time-space tradeoffs for traversing undirected graphs, using structured models based on Cook and Rackoff's "Jumping Automata for Graphs". Our three main contributions are the following.

First, we investigated the power of the model. It is easy to see that JAGs with sufficiently many pebbles can simulate well known algorithms such as depth-first search and random walk. More surprisingly, we have shown that an extremely simple variant of this model (a 2-pebble WAG, one of whose pebbles is passive) is nearly as powerful as a Turing machine. Specifically, for general undirected graph problems, it can simulate a Turing machine with only a constant factor increase in space and a polynomial factor increase in time.

Second, we have shown a lower bound on the number of states required by such machines — a WAG with one active and P passive pebbles requires $\Omega(n/P)$ states to traverse even such a simple graph as an n-cycle, independent of time. An interesting corollary is that universal traversal sequences for labeled n-cycles cannot consist solely of the repetition of some short sequence.

Finally, we have shown a strong tradeoff for graph traversal — a quadratic lower bound on the product of time and space for nondeterministic JAGs with strong jumping, one active pebble, and any number of unmovable pebbles. For example, achieving linear time requires linear space, implying that depth-first search is optimal on this model. Since our bound applies to nondeterministic algorithms for nonconnectivity, it also implies that closure under complementation of nondeterministic space-bounded complexity classes is achieved only at the expense of increased time, and that the algorithm of Feige [28] (based on Broder *et al.* [20] and Barnes and Feige [7]) cannot be made both errorless and substantially faster. We also showed that our lower bound is tight.

The obvious important problem is to strengthen and generalize these lower bounds. Following an earlier version of this paper [10], Edmonds [26] proved a time-space tradeoff on general JAGs: for every $z \geq 2$, a JAG with at most $\frac{1}{28z}\frac{\log n}{\log\log n}$ pebbles and at most $2^{\log^z n}$ states requires time $n \cdot 2^{\Omega((\log n)/(\log\log n))}$ to traverse 3-regular graphs. The ultimate goal might be to prove that $ST =$

$\Omega(mn)$ for JAGs, or even for general models of computation.

References

[1] L. M. Adleman. Two theorems on random polynomial time. In *19th Annual Symposium on Foundations of Computer Science*, pages 75–83, Ann Arbor, MI, Oct. 1978. IEEE.

[2] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. W. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *20th Annual Symposium on Foundations of Computer Science*, pages 218–223, San Juan, Puerto Rico, Oct. 1979. IEEE.

[3] N. Alon, Y. Azar, and Y. Ravid. Universal sequences for complete graphs. *Discrete Applied Mathematics*, 27:25–28, 1990.

[4] A. Bar-Noy, A. Borodin, M. Karchmer, N. Linial, and M. Werman. Bounds on universal sequences. *SIAM Journal on Computing*, 18(2):268–277, Apr. 1989.

[5] G. Barnes, J. F. Buss, W. L. Ruzzo, and B. Schieber. A sublinear space, polynomial time algorithm for directed s-t connectivity. In *Proceedings, Structure in Complexity Theory, Seventh Annual Conference*, pages 27–33, Boston, MA, June 1992. IEEE. To appear, *SIAM Journal on Computing.*

[6] G. Barnes and J. A. Edmonds. Time-space lower bounds for directed s-t connectivity on JAG models. In *Proceedings 34th Annual Symposium on Foundations of Computer Science*, pages 228–237, Palo Alto, CA, Nov. 1993. IEEE.

[7] G. Barnes and U. Feige. Short random walks on graphs. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 728–737, San Diego, CA, May 1993.

[8] G. Barnes and W. L. Ruzzo. Deterministic algorithms for undirected s-t connectivity using polynomial time and sublinear space. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, pages 43–53, New Orleans, LA, May 1991. To appear, *Computational Complexity.*

[9] P. W. Beame, A. Borodin, P. Raghavan, W. L. Ruzzo, and M. Tompa. A time-space tradeoff for undirected graph traversal by walking automata. *SIAM Journal on Computing.* To appear.

[10] P. W. Beame, A. Borodin, P. Raghavan, W. L. Ruzzo, and M. Tompa. Time-space tradeoffs for undirected graph traversal. In *Proceedings 31st Annual Symposium on Foundations of Computer Science*, pages 429–438, St. Louis, MO, Oct. 1990. IEEE.

[11] P. Berman and J. Simon. Lower bounds on graph threading by probabilistic machines. In *24th Annual Symposium on Foundations of Computer Science*, pages 304–311, Tucson, AZ, Nov. 1983. IEEE.

[12] M. Blum and D. C. 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, Ann Arbor, MI, Oct. 1978. IEEE.

[13] 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, Providence, RI, Oct. 1977. IEEE.

[14] B. Bollobás. *Extremal Graph Theory with Emphasis on Probabilistic Methods*, volume 62 of *Regional Conference Series in Mathematics.* Published for the Conference Board of the Mathematical Sciences by the American Mathematical Society, 1986.

[15] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications.* MacMillan, 1976. Revised paperback edition, 1977.

[16] A. Borodin. Structured *vs.* general models in computational complexity. *L'Enseignement Mathématique*, XXVIII(3-4):171–190, July-Dec. 1982. Also in [39, pages 47–65].

[17] A. Borodin, S. A. Cook, P. W. Dymond, W. L. Ruzzo, and M. Tompa. Two applications of inductive counting for complementation problems. *SIAM Journal on Computing*, 18(3):559–578, June 1989. See also 18(6): 1283, Dec. 1989.

[18] A. Borodin, W. L. Ruzzo, and M. Tompa. Lower bounds on the length of universal traversal sequences. *Journal of Computer and System Sciences*, 45(2):180–203, Oct. 1992.

[19] M. F. Bridgland. Universal traversal sequences for paths and cycles. *Journal of Algorithms*, 8(3):395–404, 1987.

[20] A. Z. Broder, A. R. Karlin, P. Raghavan, and E. Upfal. Trading space for time in undirected s-t connectivity. *SIAM Journal on Computing*, 23(2):324–334, Apr. 1994.

[21] J. Buss and M. Tompa. Lower bounds on universal traversal sequences based on chains of length five. *Information and Computation*, 120(2):326–329, Aug. 1995.

[22] A. K. Chandra, P. Raghavan, W. L. Ruzzo, R. Smolensky, and P. Tiwari. The electrical resistance of a graph captures its commute and cover times. *Computational Complexity.* To Appear.

[23] A. Cobham. The recognition problem for the set of perfect squares. Research Paper RC-1704, IBM Watson Research Center, 1966.

[24] S. A. Cook and C. W. Rackoff. Space lower bounds for maze threadability on restricted machines. *SIAM Journal on Computing*, 9(3):636–652, Aug. 1980.

[25] D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang. Protocol verification as a hardware design aid. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 522–525. IEEE Computer Society, 1992.

[26] J. A. Edmonds. Time-space trade-offs for undirected ST-connectivity on a JAG. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 718–727, San Diego, CA, May 1993.

[27] J. A. Edmonds and C. K. Poon. A nearly optimal time-space lower bound for directed st-connectivity on the NNJAG model. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, pages 147–156, Las Vegas, NV, May 1995.

[28] U. Feige. A randomized time-space tradeoff of $\tilde{O}(m\hat{R})$ for USTCON. In *Proceedings 34th Annual Symposium on Foundations of Computer Science*, pages 238–246, Palo Alto, CA, Nov. 1993. IEEE.

[29] P. Hall. On representatives of subsets. *J. London Math. Soc.*, 10:26–30, 1935.

[30] A. Hemmerling. *Labyrinth Problems: Labyrinth-Searching Abilities of Automata*, volume 114 of *Teubner-Texte zur Mathematik.* B. G. Teubner Verlagsgesellschaft, Leipzig, 1989.

[31] I. N. Herstein. *Topics in Algebra.* John Wiley & Sons, second edition, 1975.

[32] S. Hoory and A. Wigderson. Universal traversal sequences for expander graphs. *Information Processing Letters*, 46(2):67–69, 17 May 1993.

[33] N. Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, Oct. 1988.

[34] S. Istrail. Polynomial universal traversing sequences for cycles are constructible. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 491–503, Chicago, IL, May 1988.

[35] S. Istrail. Constructing generalized universal traversing sequences of polynomial size for graphs with small diameter. In *Proceedings 31st Annual Symposium on Foundations of Computer Science*, pages 439–448, St. Louis, MO, Oct. 1990. IEEE.

[36] J. D. Kahn, N. Linial, N. Nisan, and M. E. Saks. On the cover time of random walks on graphs. *Journal of Theoretical Probability*, 2(1):121–128, Jan. 1989.

[37] H. J. Karloff, R. Paturi, and J. Simon. Universal traversal sequences of length $n^{O(\log n)}$ for cliques. *Information Processing Letters*, 28:241–243, Aug. 1988.

[38] R. P. Kurshan. The complexity of verification. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing*, pages 365–371, Montréal, Québec, Canada, May 1994.

[39] *Logic and Algorithmic,* An International Symposium Held in Honor of Ernst Specker, Zürich, Feb. 5–11, 1980. Monographie No. 30 de L'Enseignement Mathématique, Université de Genève, 1982.

[40] M. L. Mauldin and J. R. R. Leavitt. Web-agent related research at the Center for Machine Translation. In *Proceedings of the ACM Special Interest Group on Networked Information Discovery and Retrieval (SIGNIDR-94)*, Aug. 1994.

[41] N. Nisan. $RL \subseteq SC$. *Computational Complexity*, 4(1):1–11, 1994.

[42] N. Nisan, E. Szemerédi, and A. Wigderson. Undirected connectivity in $O(\log^{1.5} n)$ space. In *Proceedings 33rd Annual Symposium on Foundations of Computer Science*, pages 24–29, Pittsburgh, PA, Oct. 1992. IEEE.

[43] N. Nisan and A. Ta-Shma. Symmetric *Logspace* is closed under complement. *Chicago Journal of Theoretical Computer Science*, 1995(1), June 1995.

[44] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[45] W. J. Savitch. Maze recognizing automata and nondeterministic tape complexity. *Journal of Computer and System Sciences*, 7(4):389–403, 1973.

[46] E. Selberg and O. Etzioni. Multi-engine search and comparison using the Metacrawler. In *World Wide Web Journal: Fourth International World Wide Web Conference Proceedings*, pages 195–208, 11–14 Dec. 1995.

[47] R. Szelepcsényi. The method of forcing for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[48] M. Tompa. Two familiar transitive closure algorithms which admit no polynomial time, sublinear space implementations. *SIAM Journal on Computing*, 11(1):130–137, Feb. 1982.

[49] M. Tompa. Lower bounds on universal traversal sequences for cycles and other low degree graphs. *SIAM Journal on Computing*, 21(6):1153–1160, Dec. 1992.

[50] D. I. Zuckerman. On the time to traverse all edges of a graph. *Information Processing Letters*, 38(6):335–337, 28 June 1991.