Time-Space Trade-Offs For Undirected ST-Connectivity on a JAG

Abstract

Undirected st-connectivity is an important problem in computing. There are algorithms for this problem that use $O(n)$ time and ones that use $O(\log n)$ space. The main result of this paper is that, in a very natural structured model, these upper bounds are not simultaneously achievable. Any probabilistic JAG requires either space $\Omega(\log^2 n/\log logn)$ or time $n^{(1+\Omega(1/\log\log n))}$ to solve undirected st-connectivity.

1 Introduction

Graph connectivity is an important problem, both practically and theoretically. Practically, it is a basic sub-routine to many graph theoretic computations. It the basic step in solving network flow optimization problems, such as project scheduling and the matching of people to jobs. Graph connectivity is also important for computer networks and search problems. Theoretically, it has been studied extensively in a number of settings. Because the undirected version of the problem is complete for symmetric log-space and the directed version is complete for non-deterministic log-space, they are natural problems for studying these classes. The study of random walks on undirected graphs and deterministic universal traversal sequences has made the problem relevant to the issue of probabilism. In addition, the directed version was used by Karchmer and Wigderson to separate monotone NC_1 from NC_2. This paper proves time-space tradeoffs for undirected st-connectivity. (They apply to the harder problem of directed st-connectivity as well.) The importance of st-connectivity is discussed in more detail in Wigderson's beautiful survey [24].

The fastest algorithms for undirected graph st-connectivity are depth-first and breadth-first search [23]. These use linear time, i.e. $O(m+n)$ for an n node, m edge graph. However, they require $\Omega(n)$ space. Alternatively, this problem can be solved deterministically in $O\left(\log^{1.5} n\right)$ space and $n^{O(1)}$ time by traversing the graph using pseudo-random generators to describe a *universal traversal sequence* [17]. If non-uniformity is allowed, these bounds can be improved to $O(\log n)$ space and $O\left(n^4 \log n\right)$ time [10,15]. If probabilism is allowed, random walks can traverse any component of the graph using $O(\log n)$ space and only $\Theta(mn)$ time [1]. More generally, Broder et al. [9] have exhibited a family of probabilistic algorithms that achieves a tradeoff between the time and the space of $S \cdot T \in m^2 \log^{O(1)} n$. This has been improved to $S \cdot T \in m^{1.5} n^{.5} \log^{O(1)} n$ [2]. A long term goal is to prove a matching lower bound.

Proving lower bounds for a general model of computation, such as a Turing machine, is beyond the reach of the current techniques. Thus it is natural to consider a "structured" model [8], whose basic operations are based on the structure of the graph, as opposed to being based on the bits in the graph's encoding. A natural structured model is the JAG ("jumping automaton for graphs") introduced by Cook and Rackoff [11]. It has a set of states and a limited supply of labeled pebbles that it can either move from a node to an adjacent node ("walk") or move directly to a node containing another pebble ("jump"). The pebbles represent node names that a structured algorithm might record in its workspace and are useful for marking certain nodes temporarily, so that they are recognizable when other pebbles reach them. Walking represents replacing a node name by the name of a node that is adjacent to it in the input graph. Jumping represents copying a previously recorded node name [5]. The space of a JAG is defined to be $S = p \log_2 n + \log_2 q$, where p is the number of pebbles and q is the number of states, because it requires $\log_2 n$ bits to store the name of a node (i.e. the location of a pebble) and $\log_2 q$ bits to record the current state.

Although the JAG model is structured, it is not weak. In particular, it is general enough so that most known algorithms for graph connectivity can be implemented on it. For example, a JAG can perform depth-first or breadth first search. It avoids cycling by leaving a pebble on each node when it first visits it. This uses $O(n \log n)$ space. Cook and Rackoff [11] show that the JAG model is powerful enough to execute an adaptation of Savitch's algorithm [20] for directed *st*-connectivity using only $O\left(\log^2 n\right)$ space. Poon [18] shows that Barnes' et al. [3] sub-linear space, polynomial time algorithm for directed *st*-connectivity runs on a JAG as well.

Furthermore, Savitch [21] shows that if one allows the JAG the additional ability to move pebbles from each node i to node $i+1$, (for an arbitrary ordering of the nodes) then the model can simulate an arbitrary Turing machine on directed graphs. Borodin, Raghavan, Ruzzo, and Tompa [5] show that even without this extra feature, JAGs can solve any undirected graph problem to within a polynomial factor as fast as Turing machines.

A number of space lower bounds have been obtained (even when an unbounded amount of time is allowed). Cook and Rackoff [11] prove a lower bound of $\Omega\left(\log^2 n / \log\log n\right)$ on the space required for a JAG to compute directed *st*-connectivity. This has been extended to randomized JAGs by Berman and Simon [6]. For undirected graph *st*-connectivity Cook and Rackoff [11] prove that $pq \in \omega(1)$ and Beame et al. [5] prove that if the pebbles are not allowed to jump, then $pq \in \Omega(n)$ even for simple 2-regular graphs. These proofs for undirected graphs show that, with a sub-linear number of states, the model goes into an infinite loop. (This method does not work when there are a linear number of states, because then the JAG is able to count the number of nodes traversed.)

Tradeoffs between the number of pebbles p used and the amount of time needed for undirected graph *st*-connectivity have also been obtained. These results are particularly strong, because they do not depend on the number of states q. A *universal traversal sequence* is simply a JAG with an unlimited number of states, but only one pebble. Borodin, Ruzzo, and Tompa [7] prove that on this model, undirected *st*-connectivity requires $\Omega\left(m^2\right)$ time. Beame, Borodin, Raghavan, Ruzzo, and Tompa [5] extend this to $\Omega\left(n^2/p\right)$ for p pebbles on 3-regular graphs with the restriction that all but one pebble are unmovable. Thus, for this very weak version of the model, a quadratic lower bound on time×space has been achieved. Beame et al. [5] also prove that there is a family of $3p$-regular undirected graphs for which *st*-connectivity with $p \in o(n)$ pebbles requires time $\Omega\left(m \log\left(\frac{n}{p}\right)\right)$, when the pebbles are unable to jump. The new result presented in this paper is the following.

Theorem 1 *Any probabilistic JAG requires either space* $\Omega(\log^2 n/\log\log n)$ *or time* $n^{(1+\Omega(1/\log\log n))}$ *to solve undirected st-connectivity even for 3-regular graphs.*

This result improves upon the previous results in at least five ways: the lower bound on **time is larger**, all pebbles are allowed to **jump**, the **degree** of the graphs considered is constant, it applies to an **average case** input instead of just the worst case input, and **probabilistic** algorithms are allowed.

The essential reason that tradeoffs arise between the time and the space required to solve a problem is that when the space is bounded the computation cannot store the results to all the previously computed subproblems and hence must recompute them over and over again. In order to prove a super linear lower bound on the time to compute *st*-connectivity, it must be proved that certain subgraphs must be traversed many times. Although the first time the JAG traverses a particular subgraph may take many time steps, it can use its states to record the structure of the subgraph so that subsequent traversals can be performed more quickly. To deal with this situation, the lower bound is actually proved on a stronger model, called the **helper JAG**. In this model, the helper, after learning the input, is allowed to set the JAG's initial state and the initial position of the pebbles in the way that minimizes the computation time. The helper is able to communicate to the JAG at least as much information about the input as it could remember from a first traversal of a subgraph. In effect, a helper JAG lower bound is a bound on the time for a regular JAG to traverse the graph subsequent times.

The helper JAG lower bound is obtained by reducing *st*-connectivity to *st*-traversal and reducing the traversal problem to a game, referred to as **the helper-parity game**. The game characterizes the relationship between the number of bits of help (or foreknowledge) and the resulting traversal time.

The paper is structured as follows. Section 2 describes the helper parity game. Section 3 formally defines the probabilistic JAG model. Section 4 reduces the *st*-traversal problem to the *st*-connectivity problem. Section 5 defines the helper JAG. Section 6 describes the fly swatter graph and gives the complexity of its traversal. Section 7 describes a line of fly swatter graphs and compares its complexity to that of the helper parity game. Section 8 describes how the input graph is recursively built. Section 9 provides the definitions of the time, pebbles, and cost used at a particular level of the recursion. This section does not provide a formal proof, but it does give some crucial intuition. Section 10 proves the main lemma by reducing the traversal problem to the helper parity game. Section 11 proves the main theorem from the main lemma. Finally, Section 12 presents some strong intuition and possible techniques for improving the lower bound to the desired bound of $n^{2-o(1)}$.

2 The Helper-Parity Game

The task of one instance of the parity game is to find a subset of the indexes on which the input vector has odd parity. This idea was introduced by Borodin, Ruzzo, and Tompa [7]. We extend this game by including a helper and multiple instances of the game. The helper is included in order to characterize what the JAG stores about a computation when a problem must be computed many times. Multiple instances of the game are included in order to decrease the number of help bits per game instances (the number of bits of information that the helper is allowed to give the player is bounded).

The **helper-parity game** with d game instances is defined as follows. There are two parties, a player and a helper. The input $\vec{\alpha}$ consists of d non-zero r bit vectors $\alpha_1, \ldots, \alpha_d \in \{0,1\}^r - \{0^r\}$, one for each of the d game instances. The helper sends the message $M(\vec{\alpha})$ to the player. It consists of a total of b bits about the d vectors. Then the player repeatedly asks the helper parity questions. A parity question specifies one of the game instances $i \in [1..d]$ and a subset of the indexes $E \subseteq [1..r]$. The answer to the parity question is the parity of the input α_i at the indexes in this set, namely $\bigoplus_{j \in E} [\alpha_i]_j$. The game is complete when the player has received an answer of 1 for each of the game instances. To simplify the game, the player is only charged one for the first question asked about a game instance and an additional one for any subsequent questions about the same game instance. Hence, the game can be thought of as the player repeatedly selecting a game instance i (in any order) and asking a single parity question E_i about that instance. If the parity is odd, the player is charged one for the game instance. Otherwise, he is charged two. Independent of the answer, the helper reveals the input α_i to the player. This is repeated until one question has been asked about each of the game instances. The cost is defined to be the average charge per game instance.

$$c_{\vec{\alpha}} = \frac{1}{d} \sum_{i \in [1..d]} \left\{ \begin{array}{ll} 1 & \text{if } \bigoplus_{j \in E_i} [\alpha_i]_j = 1 \\ 2 & \text{otherwise} \end{array} \right\}.$$

Without any help, the expected charge per game instance is 1.5. Lemma 1 below proves that if the amount of help is less than a fraction of a bit per game instance, then the player cannot do much better than this.

Defining the game so that the vector α_i is revealed after only one question is asked about it, simplifies the proof of the *st*-connectivity lower bound without significantly effecting the bound obtained.

Lemma 1 $\Pr_{\vec{\alpha}} [c_{\vec{\alpha}} < 1.5 - \epsilon] \leq 2^b e^{-(2\epsilon^2 - 2^{-r+1})d}$.

Proof of Lemma 1 [19]: Fix a helper message $m \in \{0,1\}^b$. Randomly choose an input $\vec{\alpha}$ uniformly over all the inputs in $(\{0,1\}^r)^d$. We will consider what the player's protocol is given this message and this input, even if the helper does not send this message on this input. For $i \in [1..d]$, let x_i be the indicator variable specifying whether the question about the i^{th} game instance had odd parity, i.e. $\bigoplus_{j \in E_i} [\alpha_i]_j = 1$. Clearly, these variables are independent and have probability $\frac{1}{2}$. The cost per game instance given the input $\vec{\alpha}$ and the actions of the player on message m are defined to be

$$c_{\langle \vec{\alpha}, m \rangle} = \frac{1}{d} \sum_{i \in [1..d]} \left\{ \begin{array}{ll} 1 & \text{if } x_i = 1 \\ 2 & \text{otherwise} \end{array} \right\}.$$

From this and Chernoff's bound [22], we get that

$$\Pr_{\vec{\alpha}} \left[c_{\langle \vec{\alpha}, m \rangle} < 1.5 - \epsilon \right] = \Pr \left[\sum_{i \in [1..d]} x_i \geq \left(\frac{1}{2} + \epsilon \right) d \right] \leq e^{-2\epsilon^2 d}.$$

Therefore, the cost is low for at most this fraction of vectors $\vec{\alpha} \in (\{0,1\}^r)^d$ or equivalently for at most $e^{-2\epsilon^2 d} 2^{rd}$ inputs $\vec{\alpha}$.

There are 2^b different helper messages m. For each message, the player's actions are different. Hence, the set of inputs on which the cost is low may be different for each message and in the worst case are disjoint. Hence, the number of inputs $\vec{\alpha}$ for which the cost is low when the player is given the correct help from the helper is at most $2^b e^{-2\epsilon^2 d} 2^{rd}$. The input $\vec{\alpha}$ is actually chosen randomly from $(\{0,1\}^r - \{0^r\})^d$. It follows that

$$\begin{aligned} \Pr_{\vec{\alpha}} \left[c_{\vec{\alpha}} < 1.5 - \epsilon \right] &\leq 2^b e^{-2\epsilon^2 d} \frac{2^{rd}}{\left(2^r - 1\right)^d} = 2^b e^{-2\epsilon^2 d} \left(1 - 2^{-r}\right)^{-d} \\ &\leq 2^b e^{-2\epsilon^2 d} 2^{2^{-r+1} d} \leq 2^b e^{-(2\epsilon^2 - 2^{-r+1}) d}. \quad \blacksquare \end{aligned}$$

My thesis [12] presents some surprising results about various versions of this game. It turns out that the helper must give the player quite of few bits of help before significantly decreasing the number of parity questions the player must ask when playing a single game instance. However, the same number of bits of help will decrease the number of questions asked down to only two question per game instance, no matter how many game instances are being played simultaneously. However, the helper must provide an additional bit of help per game instance to decrease the number of questions asked per game instance below $2 - \epsilon$.

This upper bound is best understood by not considering the message sent by the helper as being information about the input, but as being random bits. With a few "random" bits, the worst case complexity decreases to the point that it matches the expected complexity for a randomized protocol, which is 2 questions per game instance. The upper and lower bounds on the number of helper bits required match the work done by Impagliazzo and Zuckerman [14] on recycling random bits.

3 The Probabilistic JAG Model

A probabilistic JAG [11] is a finite automaton with p distinguishable pebbles and q states. The input to a JAG is an n vertex d regular undirected graph with two distinguished vertices s and t. For each vertex v, there is a labeling of the (half) edges emanating from v with d distinct labels. The set of labels used at different vertices is the same, but an edge can receive two different labels at its two endpoints. One of the pebbles is initially placed on the distinguished node t and other $p-1$ are placed on s.

The program of the JAG may depend non-uniformly on n and on the degree d of the graph. What the JAG does each time step depends on the current state, which pebbles coincide on the same vertices, which

pebbles are on the distinguished vertices s and t, and the value $R \in \{0,1\}^*$ of some random bits. Based on this information, the automaton changes state and does one of the following two things. It either selects some pebble $P \in [1..p]$ and some label $l \in [1..d]$ and **walks** P along the edge with label l or it selects two pebbles $P, P' \in [1..p]$ and **jumps** P to the vertex occupied by P'. A JAG that solves st-connectivity enters an accepting state if and only if there is a path from s to t in the input graph.

The space of a JAG is defined to be $S = p \log_2 n + \log_2 q$, where p is the number of pebbles and q is the number of states, because it requires $\log_2 n$ bits to store the name of a node (i.e. the location of a pebble) and $\log_2 q$ bits to record the current state.

In this paper, the running time of a deterministic algorithm averaged over inputs according to a fixed input distribution is considered instead of considering the expected running time for a probabilistic algorithm on the worst case input. According to Yao [25] this is sufficient.

4 Graph Traversal

If it is possible for a pebble of a JAG to walk from s to t on a graph, then a graph is st-connected. However, a JAG can determine that the graph is st-connected in other ways. For example, suppose that at time T_0, pebble P_s is on vertex s, P_t is on t, and P_1 and P_2 are both on some third vertex v; at time $T' \in [T_0..T_1]$, P_s and P_1 are both on the same vertex v'; and at time $T'' \in [T_0..T_1]$, P_t and P_2 are both on some other vertex v''. If these pebbles only walk along edges of the graph, then it follows that the graph is st-connected.

Additional complexity is caused by the pebbles being able to jump. A single pebble may not walk these paths from s to t. Instead, one pebble may walk part of the way. Another pebble may jump to this pebble and continue on the walk. In general, one cannot assume that a task is completed by "a specific pebble", because the pebbles are able to continually change places and each could complete some fraction of the task.

Such complex procedures for determining st-connectivity are captured by the **st-traversal** problem, which is formally defined as follows. Given a JAG computation on a graph G, the **traversal graph** H is defined as follows. For every vertex v of G and step $T \in [T_0..T_1]$, let $\langle v, T\rangle$ be a vertex of H if and only if there is a pebble on vertex v at time T. Let $\{\langle u, T\rangle, \langle v, T+1\rangle\}$ be an edge of H if a pebble walks along edge $\{u, v\}$ in G during step T and let $\{\langle v, T\rangle, \langle v, T+1\rangle\}$ be an edge of H if there is a pebble that remains on vertex v during step T. We say that the JAG **traverses** the graph G from vertex s to vertex t during the time interval $[T_0..T_1]$ if and only if there is an undirected path in H between $\langle s, T_s\rangle$ and $\langle t, T_t\rangle$ for some $T_s, T_t \in [T_0..T_1]$.

In the example given above there is a traversal path comprised of the four segments (see Figure 1): from $\langle s, T_0\rangle$ to $\langle v', T'\rangle$ following the movements of P_s, from $\langle v', T'\rangle$ to $\langle v, T_0\rangle$ following the movements of P_1 backwards in time, from $\langle v, T_0\rangle$ to $\langle v'', T''\rangle$ following the movements of P_2, and from $\langle v'', T''\rangle$ to $\langle t, T_0\rangle$ following the movements of P_t backwards in time. From the existence of this path, the JAG can deduce that s and t are connected.

The st-connectivity decision problem and the problem of traversing from s to t are closely related. Let $\mathcal{F}$ be a family of connected graphs with distinguished nodes s and t. Let $G_{s,t} \cup G_{s',t'}$ be the graph comprised of two identical disjoint copies of the graph $G \in \mathcal{F}$. Let $G_{s,t'} \cup G_{s',t}$ be the same except that one copy has the vertices s and t' and the other has s' and t. The first is st-connected and the second is not. Let $\mathcal{F}'$ be the family of graphs $\{G_{s,t} \cup G_{s',t'} \mid G \in \mathcal{F}\} \cup \{G_{s,t'} \cup G_{s',t} \mid G \in \mathcal{F}\}$.

Lemma 2 *If a JAG solves st-connectivity (in T steps) with probability $\frac{1}{2} + \epsilon$ for input graphs uniformally chosen from $\mathcal{F}'$, then a similar JAG can perform st-traversal (in T steps) with probability 2ϵ for graphs uniformally chosen from $\mathcal{F}$.*

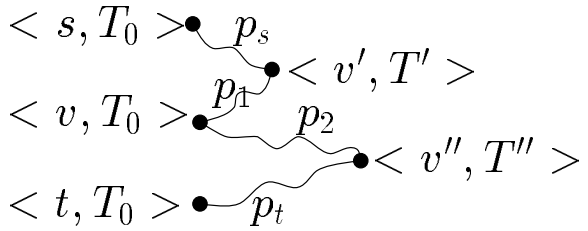

Figure 1: A path in the traversal graph of G

Proof of Lemma 2: Suppose that there is a JAG that solves st-connectivity (in T steps) with probability $\frac{1}{2} + \epsilon$ for graphs uniformally chosen from $\mathcal{F}'$. We will say that the same JAG can perform st-traversal (in T steps) on a graph $G \in \mathcal{F}$, if on input $G_{s,t} \cup G_{s',t'}$ or $G_{s,t'} \cup G_{s,t'} \in \mathcal{F}'$ it either traverses from s to t or from s' to t'. By way of contradiction, suppose for a random graph $G \in \mathcal{F}$, the probability that it does this is strictly less than 2ϵ. Consider one of the $1 - 2\epsilon$ fraction of the graphs $G_{s,t} \cup G_{s',t'}$ on which it neither traverses neither from s to t nor from s' to t'. Consider the connected components of H. At each point in time, the pebbles can be partitioned based on which connected component of H they are in. A pebble can change which part of this partition it is in by jumping, but not by traversing an edge. Since there is no path from s to t in H, the node $\langle s, T_s \rangle$, for any time step T_s, is in a different component than the nodes $\langle t, T_t \rangle$ for any time step T_t. Similarly, for all time steps $T_{s'}$ and $T_{t'}$, nodes $\langle s', T_{s'} \rangle$ and nodes $\langle t', T_{t'} \rangle$ are in different components. If this JAG was given $G_{s,t'} \cup G_{s',t}$ as input instead, the connected components of H and the partitioning of the pebbles would be isomorphic. It follows that the computation on $G_{s,t} \cup G_{s',t'}$ and on $G_{s,t'} \cup G_{s',t}$ are identical. Therefore, for at least a half of the $1 - 2\epsilon$ fraction of the graphs being considered the JAG must give the wrong answer. This contradicts the assumption. ■

By the definition of the st-traversal problem, all the pebbles are initially placed on s and t, and not within the graph. However, the proof of the lower bound uses an inductive argument for which the inductive step requires a somewhat stronger hypothesis than the main result, namely that the result holds for traversing a sub-graph (also with distinguished nodes s' and t') with an arbitrary initial placement of the pebbles and with an arbitrary initial state. If the sub-graph initially contains pebbles within the sub-graph or if the pebbles enter the sub-graph via both the s' and the t' node, then the path through the traversal graph H may go forwards and backwards in time many times, as demonstrated in the above example. However, the following defines the type of traversals for which proving a lower bound is easier. Consider a sub-graph with distinguished nodes s' and t', that initially contains no pebbles and which is built so that it can be entered by pebbles only via s' or t'. We will say that the sub-graph has been **forward traversed** from s' to t' if it is traversed, yet, during the time period of the traversal, pebbles enter the subgraph via only one of the distinguished nodes s' or t', but not both. When this occurs, there must be a path from s' to t' or from t' to s' that proceeds only forwards in time.

Consider a line of d sub-graphs, the i^{th} of which has distinguished nodes s_i and t_i, which are connected by the nodes s_i and t_{i+1} being the same node. The input graph will be many copies of this line of graphs. Consider a computation on this input graph starting with some initial placement of the pebbles and stopping when one of the lines of sub-graphs has been traversed. Because the line is traversed, each of the sub-graphs in the line must have been traversed. However, only some of these sub-graphs would have been *forward traversed*. These need to be identified. Let $S_0 \subset [1..d]$ consist of those i for which, some copy of the line initially contains a pebble in its i^{th} sub-graph. Define $S_1 \subset [1..d] - S_0$ to consist of those i such that the first time the i^{th} sub-graph in some line is traversed, it is not forward traversed. These sub-graphs do not initially contain pebbles, hence, when they are first traversed, pebbles must enter them via both the distinguished nodes s_i and t_i.

Claim 1 $|S_0 \cup S_1| \leq 2p + 1$

Proof of Claim 1: $|S_0| \leq p$, because there are only p pebbles. Consider two indices i and $i' \in S_0$, such that $i < i'$ and there is no $i'' \in S_0$ strictly between them $i < i'' < i'$. Consider two indexes j and j', such that $i < j < j' < i'$. If $j, j' \in S_1$, then pebbles would need to enter the j^{th} sub-graph via t_j and enter the j'^{th} sub-graph via $s_{j'}$. How did pebbles get in the line of sub-graphs between these two nodes without pebbles first traversing the j^{th} or the j'^{th} sub-graphs? Recall pebbles cannot jump to nodes not already containing pebbles. This is a contradiction. Hence, there can only be one $j \in S_1$ between i and i'. There can also be at most one $j \in S_1$ before first $i \in S_0$ and at most after the last. Hence, $|S_1| \leq |S_0| + 1$. ■

For all $i \in [1..d] - (S_0 \cup S_1)$, the i^{th} sub-graph is forward traversed. The parameters will be defined so that $p \in o(d)$, so that most of the sub-graphs need to be forward traversed.

5 The Helper JAG

If the goal of the JAG is to compute the st-connectivity problem, then for any given input graph, a helper could tell the JAG the answer by communicating only one bit of help. On the other hand, we will show that many bits of help are required to significantly improve the time for the JAG to actually traverse from s to t in a certain class of graphs.

Having formally defined st-traversal, we are now able to formally define a Helper JAG. It is the same as a regular JAG except that the helper, who knows the complete input, is able to set the initial state and pebble configuration in a way that minimizes the traversal time. Let $T\langle G, Q, \Pi\rangle$ be the time required for a regular JAG to traverse the input graph G starting in state $Q \in [1..q]$ and with $\Pi \in [1..n]^p$ specifying for each of the p pebbles which of the n nodes it is initially on. The time for the corresponding helper JAG to traverse G is defined to be $\min_{\langle Q,\Pi\rangle} T\langle G, Q, \Pi\rangle$. It is often easier to think of the helper giving the JAG $b = \log(qn^p)$ bits of information.

6 A Fly Swatter Graph

The basic components of the input graphs defined in Section 8 are the **fly swatter graph** (see Figure 2). A fly swatter graph consists of two identical graphs with r **switches** between them. It is very similar to the squirrel cage graph defined in [5]. Each half consists of a path of length $\frac{h}{2}$, called the **handle**, and a swatting part. The swatting part consists of two parallel paths of length $r+1$ that are both connected to one end of the handle. The distinguished nodes s and t are located at the ends of the handles furthest from the swatting parts. Suppose the swatting part of one half of the graph contains the paths $u_0^0, u_0'^0, u_1^0, u_1'^0, \ldots, u_{r-1}'^0, u_r^0$ and $v_0^0, v_0'^0, v_1^0, v_1'^0, \ldots, v_{r-1}'^0, v_r^0$ and the swatting part of the other half contains the paths $u_0^1, u_0'^1, u_1^1, u_1'^1, \ldots, u_{r-1}'^1, u_r^1$ and $v_0^1, v_0'^1, v_1^1, v_1'^1, \ldots, v_{r-1}'^1, v_r^1$. Then the setting of the switches between the halves is specified by a non-zero vector $\alpha \in \{0,1\}^r$ as follows. For each $j \in [1..r]$, the j^{th} switch consists of the two **cross-over edges** $\{u_j^0, v_j^{[\alpha]_j}\}$ and $\{u_j^1, v^{\overline{[\alpha]_j}}\}$. Note that if $[\alpha]_j = 0$, then the switch remains within the same half and if $[\alpha]_j = 1$, then the switch crosses over from one half to the other. (The notation $[\alpha]_j$ is used to denote the j^{th} bit of the vector α. The notation α_i is reserved to mean the i^{th} vector in a vector of vectors.) The extra nodes $u_i'^0$ and $v_i'^0$ are added so that the smallest square containing cross-over edges contains six edges.

Forward traversing from s to t in the fly swatter specified by α requires traversing a sequence of switches $E \in [1..r]^*$ for which the parity of the bits of α on the indexes in E is 1, i.e. $\bigoplus_{j \in E} [\alpha]_j = 1$. To be able complete this task, the JAG must be able to determine the parity of such a sequence E. There are two ways in which the JAG can ask a **parity question**. The lower bound will prove that these are the only ways in which the JAG is able to acquire the information about the input.

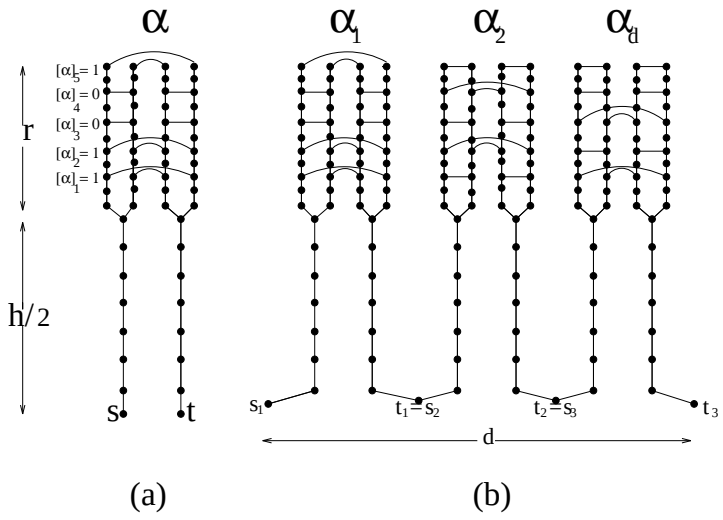

Figure 2: A fly swatter graph and a line of fly swatters

The first method of asking a parity question requires only one pebble, but a great deal of time. The pebble enters the fly swatter via the distinguished node s (or t), traverses up the handle, through a sequence of switches $E \in [1..r]^+$, and back down the handle. While the pebble is inside the fly swatter, the JAG has no way of learning which half the pebble is in, because the two halves are indistinguishable. However, when the pebble reaches the bottom of the handle, the parity of the sequence is determined by whether the distinguished node s or t is reached. Each handle contains $h/2$ edges. Therefore, asking a parity question with one pebble requires the JAG to traverse at least h edges. This is illustrated in Figure 3 (a).

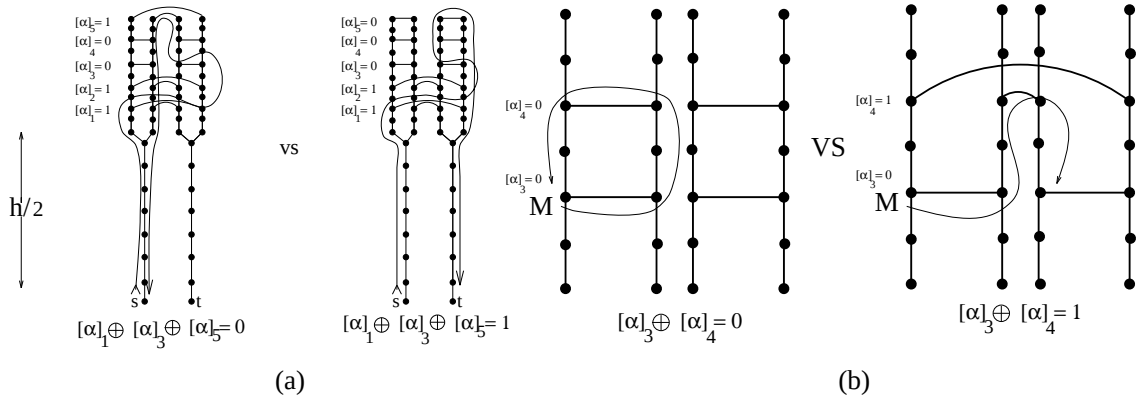

Figure 3: A parity question without and with a marker

The second method requires two pebbles, one of which acts as a **marker** and the other of which traverses a sequence of switches E. The parity of the sequence is determined by whether the traversing pebble returns to the marker. For example, if a pebble starts at node u_2^0 and walks the edges labeled $\langle switch, up, switch, down \rangle$, then one possible sequence of nodes for the pebble to follow is $u_2^0, v_2^0, v_3^0, u_3^0, u_2^0$ and another is $u_2^0, v_2^0, v_3^0, u_3^1, u_2^1$ depending on which switches in the graph are switched. Provided the JAG leaves a pebble as a marker at the node u_2^0, it can differentiate between these two sequences and learn whether $[\alpha]_2 \oplus [\alpha]_3$ is 0 or 1. This is illustrated in Figure 3 (b). Even though a parity question E (for example, the parity of all the bits in α) may require $\Theta(r)$ edges to be traversed, the lower bound will only charge the JAG for the traversal of at most two edges for a parity question using a marker.

If the JAG can only gain information about the input by asking parity questions in these two ways, then a JAG that solves st-connectivity for a fly swatter graph must be able to solve the following version of the parity game: The input to this game consists of a single non-zero vector $\alpha \in \{0,1\}^r$. The player, after receiving no help, asks parity questions. A parity question specifies a subset of the indexes $E \subseteq [1..r]$. The answer to the parity question is the parity of the input α at the indexes in this set, namely $\bigoplus_{j \in E} [\alpha]_j$. The complexity is the number of questions asked before the player asks a parity question with answer 1.

Beame et al. [5] prove that, for this game, r questions must be asked in the worst case. The proof uses the fact that $\alpha \in \{0,1\}^r - \{0^r\}$ forms a vector space of dimension r. In this way, they prove that for one pebble, $\Theta(hr) = \Theta(n^2)$ time is needed. However, we are considering more than one pebble. With two pebbles the JAG can traverse the fly swatter graph in linear time.

In order to prove lower bounds when the JAG has more than one pebble, a more complex graph is needed. The fly swatter graph will be a subgraph of this more complex graph. In order to traverse this complex graph the JAG will have to traverse a particular fly swatter subgraph many times. Hence, on subsequent traversals of this fly swatter the JAG may have some precomputed information about it. This is modeled by a helper providing the JAG with this precomputed information.

7 The Helper and a Line of Fly Swatter Graphs

The helper communicates information to the JAG about the input graph by specifying the initial state $Q \in [1..q]$ and location of each of the p pebbles. Hence, the amount of information that the helper is able to provide is limited to at most $b = \log(qn^p)$ bits. Only $\log r << b$ bits are required for the JAG to be able to traverse a fly swatter in linear time. However, b is not enough bits of help to simultaneously provide sufficient information about many fly swatter graphs. For this reason, we require the JAG to traverse a **line of d fly swatters**. Such a line is specified by the parameters $r \in O(1)$, $h \in O(1)$, $d \in \log^{O(1)} n$ and the vector of vectors $\vec{\alpha} = \langle \alpha_1, \ldots, \alpha_d \rangle \in (\{0,1\}^r - \{0^r\})^d$. The d graphs are connected by making the distinguished nodes t_i and s_{i+1} be the same node, for $i \in [1..d-1]$. This is illustrated in Figure 2 (b).

The similarities between the parity game and the traversal of a line of fly swatter graphs should be clear, at least informally. Lemma 1 proves that if the JAG is only able to gain information by asking parity questions and $b << d$, then the average number of questions the JAG must ask is $(1.5-\epsilon)d$. Therefore, if a marker is utilized in none of the questions, then the number of edges traversed by the JAG is $h(1.5-\epsilon)d$ and, if a marker is utilized in all of the questions then $2(1.5-\epsilon)d$ edges are traversed. Note that without a marker, the time required is roughly a factor of $(1.5-\epsilon)$ larger than the number of edges. This factor is not very impressive, but its effect is magnified in a recursive construction.

8 The Recursive Fly Swatter Graphs

Let $G(\vec{\alpha}_l)$ denote the line of fly swatters specified by the vector of vectors $\vec{\alpha}_l = \langle \alpha_{\langle l,1 \rangle}, \ldots, \alpha_{\langle l,d \rangle} \rangle \in (\{0,1\}^r - \{0^r\})^d$. For each $l \geq 0$, we recursively define a graph. Define $G(\emptyset)$ to be a single edge. Define $G(\vec{\alpha}_1, ..., \vec{\alpha}_l)$ to be the graph $G(\vec{\alpha}_l)$ where each edge is replaced by a super edge consisting of a copy of $G(\vec{\alpha}_1, ..., \vec{\alpha}_{l-1})$. The node $s_{\langle l-1,1 \rangle}$ is one end of the super edge and the node $t_{\langle l-1,d \rangle}$ is the other end. All the super edges in one level are the same. The family of recursive fly swatter graphs is $\{ G(\vec{\alpha}_1, ..., \vec{\alpha}_L) \mid \vec{\alpha}_1, \ldots, \vec{\alpha}_L \in (\{0,1\}^r - \{0^r\})^d \}$, where L is such that n is the total number of nodes. (Gadgets can be added to make the graph 3-regular without changing it significantly). The graph $G(\vec{\alpha}_1, ..., \vec{\alpha}_l)$ contains $(h+10r)d$ copies of $G(\vec{\alpha}_1, ..., \vec{\alpha}_{l-1})$. Therefore, the total number of edges is at most $[(h+10r)d]^L$. The number of nodes n is approximately two thirds of the number of edges.

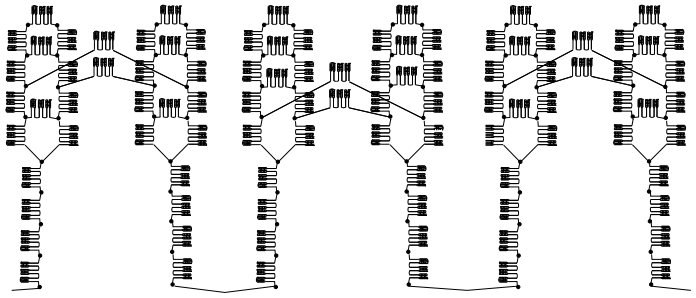

Figure 4: The recursive line of fly swatters graph (Sorry, the cycles in the figure have length 4 instead of 6.)

A crucial observation in understanding the complexity of traversing this graph is that a pebble can only be used as a marker in one recursion level at a time. To demonstrate this, consider $L = 2$ levels of recursion and $p = 2$ pebbles. If a parity question is asked about the top level vector $\vec{\alpha}_L$ by leaving a pebble as a marker, then only one pebble remains to traverse the sequence of super edges required by the question. Hence, these super edges, which are themselves lines of fly swatters, must be traversed with the use of only one pebble. Alternatively, if two pebbles are used to traverse each super edge, then there is "effectively" one pebble for the traversal of the top level. See Figures 3 (b) and 5.

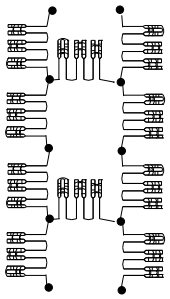

Figure 5: Ten super edges of a two level recursive fly swatter graph

An intuitive explanation of the lower bound can now be given. (A formal proof is provided in Section 11.) There are p pebbles, hence at most $p-1$ markers. It follows that $L-p+1$ levels are traversed without the use of a marker. Note, as well, that the time to traverse a copy of $G(\vec{\alpha}_1, ..., \vec{\alpha}_l)$ is the number of super edges traversed in the l^{th} level subgraph $G(\vec{\alpha}_l)$ multiplied by the time to traverse each super edge $G(\vec{\alpha}_1, ..., \vec{\alpha}_{l-1})$. Therefore, an estimation of the total time is the product of the number of super edges traversed at each of the L levels,

$$\begin{aligned} T &\geq [h(1.5-\epsilon)d]^{L-p+1}[(1.5-\epsilon)d]^{p-1} \\ &= (1.5-\epsilon)^L \times (hd)^L \times h^{-p+1} \end{aligned} \tag{1}$$

The parameters are chosen as follows: $r, h \in \Theta(1)$, $d \in \log^{\Theta(1)} n$, and $L \in \Theta\left(\frac{\log n}{\log\log n}\right)$. Then the first factor becomes $2^{\Omega(\frac{\log n}{\log\log n})}$, the second is close to n (assuming $r << h$), and the third is insignificant compared to the first assuming that p is a constant fraction of $\frac{L}{\log h} \in \Theta\left(\frac{\log n}{\log\log n}\right)$. This is the bound claimed in Theorem 1.

9 The Time, Pebbles, and Cost used at Level l

The lower bound is proved by induction on the number of levels of recursion l. For each $l \in [1..L]$, we prove a lower bound on the cost to traverse some copy of $G(\vec{\alpha}_1, ..., \vec{\alpha}_l)$. Define $\vec{\gamma} = \langle \vec{\alpha}_1, \ldots, \vec{\alpha}_{l-1} \rangle$ and $\vec{\beta} = \langle \vec{\alpha}_{l+1}, \ldots, \vec{\alpha}_L \rangle$ so that $G(\vec{\alpha}_1, ..., \vec{\alpha}_L)$ and $G\left(\vec{\gamma}, \vec{\alpha}_l, \vec{\beta}\right)$ denote the same graph. Think of $G(\vec{\gamma}, \vec{\alpha}_l)$ (the sub-graph traversed) as a line of d fly swatters $G(\vec{\alpha}_l)$ with each of its super edges being a copy of $G(\vec{\gamma})$. The super edge $G(\vec{\gamma})$ does not need to be understood, because the induction hypothesis proves a lower bound on the time to traverse it. In the graph $G\left(\vec{\gamma}, \vec{\alpha}_l, \vec{\beta}\right)$, there are many copies of $G(\vec{\gamma}, \vec{\alpha}_l)$. The graph $G(\vec{\beta})$ is called the **context** in which these copies of $G(\vec{\gamma}, \vec{\alpha}_l)$ appear.

Informally, the time required to traverse $G(\vec{\gamma}, \vec{\alpha}_l)$ is the number of super edges of $G(\vec{\alpha}_l)$ traversed multiplied by the time to traverse a super edge $G(\vec{\gamma})$. This fails to be true, because each traversal of a super edge may require a different amount of time. This difference in time is caused by the number of markers (pebbles) beings used in the traversal and by the state and position of the pebbles before the traversal. Note, differences in traversal times are not caused by differences in the structure of the super edges, because they are all identical.

Let $Q \in [1..q]$ be a state of the JAG and let $\Pi \in [1..n]^p$ specify which node of $G\left(\vec{\gamma}, \vec{\alpha}_l, \vec{\beta}\right)$ each of the p pebbles is on. Consider the JAG computation starting in the configuration described by Q and Π on the graph $G\left(\vec{\gamma}, \vec{\alpha}_l, \vec{\beta}\right)$ until some copy of $G(\vec{\gamma}, \vec{\alpha}_l)$ is traversed. Define $T\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle, Q, \Pi\right]$ to be the number of time steps taken. This will be abbreviated to $T[l]$ when the rest of the parameters are understood. Define $p\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle, Q, \Pi\right]$ (abbreviated to $p[l]$) to be

$$p\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle, Q, \Pi\right] \quad = \quad \max_{T \in [1..T[l]]} \left[p + 1 - (\#\text{ copies of } G(\vec{\gamma}, \vec{\alpha}_l) \text{ containing pebbles at time } T)\right].$$

At no time during the interval does a copy of $G(\vec{\gamma}, \vec{\alpha}_l)$ contain more than $p[l]$ pebbles. For example, suppose there are two copies of $G(\vec{\gamma}, \vec{\alpha}_l)$ containing pebbles. One copy contains at least one pebble and, therefore, the other copy contains no more than $p + 1 - 2 = p - 1$ pebbles. Think of the "(# copies of $G(\vec{\gamma}, \vec{\alpha}_l)$ containing pebbles)" as the number of pebbles being used as "markers" in the graph $G(\vec{\beta})$. Essentially, no more than one of these pebbles is available to be used in the traversal of $G(\vec{\gamma}, \vec{\alpha}_l)$.

Define the **cost** incurred by the JAG in traversing a copy of $G(\vec{\gamma}, \vec{\alpha}_l)$ to be

$$w\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right] \quad = \quad \min_{\langle Q, \Pi \rangle} h^{p\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle, Q, \Pi\right]} T\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle, Q, \Pi\right]$$

The motivation for using $h^{p[l]}T[l]$ comes from Equation 1. Since the average time $T[l]$ to traverse $G(\vec{\gamma}, \vec{\alpha}_l)$ can be estimated by $(1.5 - \epsilon)^l (hd)^l h^{-p[l]+1}$, the quantity $h^{p[l]}T[l]$ is essentially independent of the number of pebbles used. The reason for minimizing over $\langle Q, \Pi \rangle$ is that we are assuming the helper sets the initial JAG configuration in a way that minimizes the cost of the traversal. The actual bounds obtained (with high probability) are the following.

$$\begin{aligned} W[0] &= h \\ W[l] &= W[l-1] \times h \times (d(1.5 - \epsilon) - 4p - 2) \\ &= \left[h \times (d(1.5 - \epsilon) - 4p - 2)\right]^l h. \end{aligned}$$

The remaining goal is to prove that with high probability (over randomly chosen inputs) the cost $w\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right]$ to traverse a copy of $G(\vec{\gamma}, \vec{\alpha}_l)$ is at least $W[l]$. Lemma 3 provides the inductive step for this proof.

10 Reducing the Helper Parity Game to *st*-Traversal

Suppose that there is a JAG algorithm for which the time to traverse a copy of $G(\vec{\gamma}, \vec{\alpha}_l)$ is only a small factor more than the time to traverse a copy of $G(\vec{\gamma})$. This means that the JAG is able to traverse the line of fly swatters $G(\vec{\alpha}_l)$ without traversing many of its super edges and hence without "asking" many parity questions about $\vec{\alpha}_l$. In effect, the JAG is able to play the helper parity game with parameters r, d, and $b = \log(qn^p)$, where r and d are the parameters defining the line of fly swatters and $\log(qn^p)$ is the space allocated to the JAG. This is captured in the following lemma.

Lemma 3 *Given an algorithm for st-traversal whose cost on input* $G\left(\vec{\gamma}, \vec{\alpha}_l, \vec{\beta}\right)$ *to traverse a subgraph at the* $l-1^{st}$ *and the* l^{th} *levels are* $w\left[l-1, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right]$ *and* $w\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right]$*, we can produce a protocol for the helper parity game for which the number of questions asked per game instance on input* $\vec{\alpha}_l$ *is* $c_{\vec{\alpha}_l}$*, such that*

$$\Pr_{\vec{\alpha}_l}\left[c_{\vec{\alpha}_l} < 1.5-\epsilon\right] \;\geq\; \Pr_{\left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle}\left[w\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right] < W[l]\right] - \Pr_{\left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle}\left[w\left[l-1, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right] < W[l-1]\right].$$

Corollary 2 *Given a helper-JAG algorithm for st-traversal whose traversal time on input* $G(\vec{\alpha}_1, ..., \vec{\alpha}_L)$ *is* $T_{\langle \vec{\alpha}_1, \ldots, \vec{\alpha}_L \rangle}$ *and the helper parity game defined in Lemma 3,*

$$\Pr_{\langle \vec{\alpha}_1, \ldots, \vec{\alpha}_L \rangle}\left[T_{\langle \vec{\alpha}_1, \ldots, \vec{\alpha}_L \rangle} < \left[h \times (d(1.5-\epsilon) - 4p - 2)\right]^L h^{-p+1}\right] \;\leq\; L \times \Pr_{\vec{\alpha}}\left[c_{\vec{\alpha}} < 1.5-\epsilon\right]$$

Proof of Corollary 2: The time and the cost incurred by the JAG to traverse the entire input graph are defined to be $T_{\langle \vec{\alpha}_1, \ldots, \vec{\alpha}_L \rangle} = T[L]$ and $w[L] = \min_{\langle Q, \Pi \rangle} h^{p[L]} T[L]$. The number of pebbles $p[L]$ used in the traversal is at most the number of pebbles p the JAG has. It follows that if traversal time is small, i.e. $T[L] < W[L]h^{-p}$, then the cost is small, i.e. $w[L] = h^{p[L]}T[L] < W[L]$. It remains to prove that for $l \leq L$,

$$\Pr_{\langle \vec{\alpha}_1, \ldots, \vec{\alpha}_L \rangle}\left[w\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right] < W[l]\right] \;\leq\; l \times \Pr_{\vec{\alpha}}\left[c_{\vec{\alpha}} < 1.5-\epsilon\right]$$

The proof proceeds by induction on l. For the base case, $l = 0$, the subgraph $G()$ is a single edge requiring at least 1 time step for traversal by at least one pebble. This guarantees that with probability 1, $w[0] = \min_{\langle Q, \Pi \rangle} h^{p[0]} T[0] \geq h = W[0]$. The induction step follows easily using Lemma 3. ■

Proof of Lemma 3: Consider a fixed algorithm for *st*-traversal. In the helper-parity protocol defined below, the game-helper learns what help to send and the game-player learns what questions to ask by running this fixed JAG algorithm as it traverses a line of fly swatters at the l^{th} level.

Both the *st*-traversal problem and the helper-parity game have the vector $\vec{\alpha}_l = \left\langle \alpha_{\langle l,1 \rangle}, \ldots, \alpha_{\langle l,d \rangle} \right\rangle \in (\{0,1\}^r - \{0^r\})^d$ as part of its input. However, the *st*-traversal problem has the additional inputs $\vec{\gamma}$ and $\vec{\beta}$. Therefore these vectors are fixed to $\vec{\gamma}'$ and $\vec{\beta}'$ in a way that satisfies the property

$$\begin{aligned}
&\Pr_{\vec{\alpha}_l}\left[w\left[l, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle\right] < W[l]\right] - \Pr_{\vec{\alpha}_l}\left[w\left[l-1, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle\right] < W[l-1]\right] \\
&\quad\leq\; \Pr_{\left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle}\left[w\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right] < W[l]\right] - \Pr_{\left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle}\left[w\left[l-1, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right] < W[l-1]\right].
\end{aligned}$$

These vectors $\vec{\gamma}'$ and $\vec{\beta}'$ are known in advance to both the game-helper and the game-player.

The first thing that the game protocol must specify is the message $M(\vec{\alpha}_l)$ sent by the game-helper on each input $\vec{\alpha}_l$. This message is defined to be $\left\langle Q_{\langle l, \vec{\alpha}_l \rangle}, \Pi_{\langle l, \vec{\alpha}_l \rangle} \right\rangle$, which specifies the configuration in which the JAG-helper initially places the JAG when $G\left(\vec{\gamma}', \vec{\alpha}_l, \vec{\beta}'\right)$ is the JAG's input graph. Note that the game-helper only sends $\log(qn^p)$ bits, because this is the number of bits needed to encode a state and the locations of all p pebbles.

The game-player learns which questions to ask the helper by simulating the JAG algorithm on the graph $G\left(\vec{\gamma}', ?, \vec{\beta}'\right)$ starting in the configuration $\left\langle Q_{\langle l,\vec{\alpha}_l\rangle}, \Pi_{\langle l,\vec{\alpha}_l\rangle}\right\rangle$. The only thing preventing the game-player from running the JAG is that he does not know the vector $\vec{\alpha}_l$. However, he can run the JAG as long as the computation does not depend on this unknown information. Specifically, suppose during the simulation, pebbles enter a fly swatter defined by a $\alpha_{\langle l,i\rangle}$ that is not known by the game-player. The game-player will be able to continue running the JAG for quite a while. However, as soon as the computation depends on which cross-over edges of the fly swatter are switched, he must stop the simulation. He then asks the game-helper a question about the game instance $\alpha_{\langle l,i\rangle}$. By definition of the parity game, the game-helper reveals to him the entire vector $\alpha_{\langle l,i\rangle}$. With this new information, the game-player is able to continue the simulation until the next such event occurs.

As done in Section 4, let $S_0 \subset [1..d]$ consist of those i for which some copy of $G(\vec{\gamma}', \vec{\alpha}_l)$ initially (i.e. according to $\Pi_{\langle l,\vec{\alpha}_l\rangle}$) contains a pebble in its i^{th} fly swatter. Note that the p pebbles of the JAG might be contained in different copies of $G(\vec{\gamma}', \vec{\alpha}_l)$, but all such copies are considered. The game-player begins the game by asking an arbitrary parity question E_i about $\alpha_{\langle l,i\rangle}$ for each $i \in S_0$.

The game-player then starts simulating the JAG. Because he knows $\alpha_{\langle l,i\rangle}$ for every fly swatter containing pebbles, he can run the JAG at least until a pebble moves into an adjacent fly swatter. The JAG might alternately move pebbles contained in different copies of $G(\vec{\gamma}', \vec{\alpha}_l)$. However, we will count the number of time steps taken in each copy separately.

If the i^{th} fly swatter in some copy of $G(\vec{\gamma}', \vec{\alpha}_l)$ is entered via both its s_i and t_i distinguished nodes, then the game-player will also ask an arbitrary parity question E_i about $\alpha_{\langle l,i\rangle}$, as long as a question has not been asked about $\alpha_{\langle l,i\rangle}$ already. The indexes for which this happens forms the set S_1, as defined in Section 4. By Claim 1, $|S_0 \cup S_1| \leq 2p+1$. Therefore, this completes at most $2p+1$ of the d game instances $\alpha_{\langle l,1\rangle}, \ldots, \alpha_{\langle l,d\rangle}$. The remaining fly swatters indexed by $i \in [1..d] - (S_0 \cup S_1)$ are *forward traversed*.

Two events will now be defined. If one of these events occurs within the i^{th} fly swatter in some copy of $G(\vec{\gamma}', \vec{\alpha}_l)$, then the game-player asks a question about the i^{th} game instance $\alpha_{\langle l,i\rangle}$. Below, we prove that the computation of the JAG through the i^{th} fly swatter does not depend on $\alpha_{\langle l,i\rangle}$ until one of these events occurs. It follows that the game-player is always capable of running the simulation and hence of executing the parity-game protocol currently being defined. We also prove that, because fly swatters indexed by $i \in [1..d] - (S_0 \cup S_1)$ are *forward traversed*, one of the events eventually occurs within each of them. From this, it follows that the parity-game protocol will terminate, asking a question about each game instance. Finally, I prove that traversing a fly swatter is at least twice as costly for the JAG when the question asked has even parity. Hence, the total cost for the parity game is proportional to the total cost for the JAG traversal.

Before continuing, recall that Section 6 described two ways to ask a parity question. The first event will be defined so that it occurs within a fly swatter when the two pebble method is used within it, i.e. a pebble is left as a marker while another pebble walks along a sequence of super edges. Similarly, the second event will be defined so that it occurs when the one pebble method is used, i.e. a pebble walks up the handle, through a sequence of switches and back down a handle again.

The first of these events is formally defined to occur within a fly swatter after the following has occurred twice during disjoint intervals of time. What must occur twice is that one of the super edges of the fly swatter is traversed and during the entire time of its traversal, there is a pebble (not necessarily the same pebble at each time step) contained in the copy of $G(\vec{\gamma}', \vec{\alpha}_l)$ containing the fly swatter, but not contained in the super edge in question. If this occurs in the i^{th} fly swatter in some copy of $G(\vec{\gamma}', \vec{\alpha}_l)$, then the player asks an arbitrary question E_i about $\alpha_{\langle l,i\rangle}$.

The second event is defined to occur within the i^{th} fly swatter, if it is initially empty, pebbles traverse up the handle, reaching the cross-over edges, and then traverse down the handle again, reaching one of the distinguished nodes $s_{\langle l,i\rangle}$ or $t_{\langle l,i\rangle}$, yet during this entire time the first event never occurs within the i^{th} fly

swatter. We claim that for this event to occur, for $i \in [1..d]-(S_0 \cup S_1)$, all the pebbles within this fly swatter must traverse a single well defined sequence of switches. When the second event occurs within the i^{th} fly swatter in some copy of $G(\vec{\gamma}', \vec{\alpha}_l)$, the game-player asks the question E_i that contains j iff the j^{th} switch was traversed an odd number of times.

Now we will prove the claim. As stated, the pebbles are initially outside of the fly swatter. Because $i \notin S_1$, pebbles do not enter the fly swatter via both the distinguished nodes $s_{\langle l,i \rangle}$ and $t_{\langle l,i \rangle}$. Without loss generality assume that they enter via $s_{\langle l,i \rangle}$. Furthermore, there are never two pebbles within the fly swatter that have four or more full super edges between them. The reason is as follows. The pebbles contained in the fly swatter are initially together, because they enter only via $s_{\langle l,i \rangle}$. In order for two pebbles to get three full super edges between them, a super edge must be traversed while there is a pebble contained in this copy of $G(\vec{\gamma}', \vec{\alpha}_l)$, but not contained in the super edge in question. For the first event to occur, this must happen twice during disjoint intervals of time. For two pebbles to have four full super edges between, a second super edge must be traversed in this way. These two traversals occur during disjoint intervals in time and hence the first event occurs. Because the first event does not occur, no two pebbles in the fly swatter ever have four full super edges between them. Hence, the pebbles must traverse up the handle more or less together. They cannot traverse in opposite directions around a square of six super edges, hence must traverse the same sequence of switches. Finally, they must traverse down the handle together. This proves the claim.

We will now prove that the game-player always has enough information to continue running the JAG. Because of the game-helper's message and the fact that $\vec{\gamma}'$ and $\vec{\beta}'$ are fixed, the only information that the player is lacking is $\vec{\alpha}_l$. In addition, $\alpha_{\langle l,i \rangle}$ is revealed as soon he asks a question about it. Therefore, the only concern is whether the game-player, even though it does not know $\alpha_{\langle l,i \rangle}$, can run the JAG as it traverses the i^{th} fly swatter, at least until one of the two events happens. As said, if the first event has not occurred, then all the pebbles must traverse the same sequence of switches. Therefore, the only influence that $\alpha_{\langle l,i \rangle}$ (i.e. which switchable edges are switched) has on the computation is which of the two fly swatter halves these pebbles are contained in. However, the JAG has no way of knowing which half the pebbles are in, because the super edges in the two halves, the structure of the halves, and even the edge labels are identical. Therefore, the player knows as much as the JAG knows (i.e. the state and the partition of the pebbles) until second event occurs.

It has now been proven that the above protocol is well defined and meets the requirements of the game. The remaining task is to prove that the cost to the parity game is proportional to the cost of the JAG's traversal. Let us first consider the cost of the JAG's traversal. The JAG is run on the input graph $G\left(\vec{\gamma}', \vec{\alpha}_l, \vec{\beta}'\right)$, starting in the configuration $\left\langle Q_{\langle l,\vec{\alpha}_l \rangle}, \Pi_{\langle l,\vec{\alpha}_l \rangle} \right\rangle$ specified by the JAG-helper, until some copy of $G(\vec{\gamma}', \vec{\alpha}_l)$ (a line of fly swatters at the l^{th} level) is traversed. By definition, the time of this traversal is $T\left[l, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle, Q_{\langle l,\vec{\alpha}_l \rangle}, \Pi_{\langle l,\vec{\alpha}_l \rangle}\right]$, the number of pebbles "used" during this traversal is $p\left[l, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle, Q_{\langle l,\vec{\alpha}_l \rangle}, \Pi_{\langle l,\vec{\alpha}_l \rangle}\right]$, and the cost incurred by the JAG is $w\left[l, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle\right] = \min_{\langle Q,\Pi \rangle} h^{p\left[l, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle, Q, \Pi\right]} T\left[l, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle, Q, \Pi\right]$. Abbreviate these values to $T[l, \vec{\alpha}_l]$, $p[l, \vec{\alpha}_l]$ and $w[l, \vec{\alpha}_l]$. For each $i \in [1..d] - S_0$, define $T[l, \vec{\alpha}_l, i]$ to be the number of time steps for the event to occur within the i^{th} fly swatter of some copy of $G(\vec{\gamma}', \vec{\alpha}_l)$. This time is at least the product of the number of super edges traversed and the minimum time to traverse a super edge on this input graph. The number of super edges traversed depends on whether a pebble was left as a marker within the fly swatter and whether a path from $s_{\langle l,i \rangle}$ to $t_{\langle l,i \rangle}$ was found in the first attempt, i.e. whether the answer to the parity was odd. The minimum cost to traverse a super edge $G(\vec{\gamma}')$ when the input graph is $G\left(\vec{\gamma}', \vec{\alpha}_l, \vec{\beta}'\right)$ is defined to be $w\left[l-1, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle\right]$. Each traversal of a super edge will cost at least this minimum. Abbreviate this with $w[l-1, \vec{\alpha}_l]$. The following claim bounds the time $T[l, \vec{\alpha}_l, i]$ for the event to occur within the i^{th} fly swatter.

Claim 2 *For each* $i \in [1..d] - (S_0 \cup S_1)$,

$$T[l, \vec{\alpha}_l, i] \geq \left\{ \begin{array}{ll} 1 & \text{if } \bigoplus_{j \in E_i} \left[\alpha_{\langle l,i \rangle}\right]_j = 1 \\ 2 & \text{otherwise} \end{array} \right\} w[l-1, \vec{\alpha}_l] h^{-p\left[l, \vec{\alpha}_l\right]+1}.$$

Proof of Claim 2:
Case 1: Suppose the first event occurs, i.e. two super edges at level $l-1$ are traversed by a pebble and during the entire time of their traversal, there is another pebble contained in the same copy of $G(\vec{\gamma}', \vec{\alpha}_l)$, but not contained in these two super edges. Consider one of these two super edges traversed and let $\left\langle Q_{\langle l-1, \vec{\alpha}_l \rangle}, \Pi_{\langle l-1, \vec{\alpha}_l \rangle} \right\rangle$ be the configuration of the JAG at the beginning of this traversal. The number of time steps, starting in this configuration, until some copy of $G(\vec{\gamma}')$ is traversed (clearly the super edge in question) is defined to be $T\left[l-1, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle, Q_{\langle l-1, \vec{\alpha}_l \rangle}, \Pi_{\langle l-1, \vec{\alpha}_l \rangle}\right]$ and the number of pebbles "used" in this traversal is defined to be $p\left[l-1, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle, Q_{\langle l-1, \vec{\alpha}_l \rangle}, \Pi_{\langle l-1, \vec{\alpha}_l \rangle}\right]$. Abbreviate these to $T[l-1, \vec{\alpha}_l]$ and $p[l-1, \vec{\alpha}_l]$. It will also be useful to use $T'[l-1, \vec{\alpha}_l]$ to denote the time interval during which this super edge is traversed and to use $T'[l, \vec{\alpha}_l]$ to denote the time interval during which the entire line of fly swatters $G(\vec{\gamma}', \vec{\alpha}_l)$ is traversed. The "cost" of the traversal of the super edge is defined to be

$$h^{p[l-1, \vec{\alpha}_l]} T[l-1, \vec{\alpha}_l].$$

This is at least

$$w[l-1, \vec{\alpha}_l] \quad = \quad \min_{\langle Q, \Pi \rangle} h^{p\left[l-1, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle, Q, \Pi\right]} \; T\left[l-1, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle, Q, \Pi\right].$$

which is the minimal cost of traversing any super edge when the helper has pre-set the JAG configuration $\langle Q, \Pi \rangle$ to minimize the cost. Solving for the traversal time gives

$$T[l-1, \vec{\alpha}_l] \quad \geq \quad w[l-1, \vec{\alpha}_l] h^{-p\left[l-1, \vec{\alpha}_l\right]}.$$

The next step is to bound the number of pebbles $p[l-1, \vec{\alpha}_l]$ "used" to traverse this super edge. The intuition is as follows. The JAG has $p[l, \vec{\alpha}_l]$ pebbles available to traverse a copy of $G(\vec{\gamma}', \vec{\alpha}_l)$. If it leaves a pebble as a marker in the l^{th} level and traverses a sequence of switchable edges with the other $p[l, \vec{\alpha}_l] - 1$ pebbles, then only $p[l, \vec{\alpha}_l] - 1$ pebbles are available for the traversal of these super edges $G(\vec{\gamma}')$. More formally, we want to prove that $p[l-1, \vec{\alpha}_l] \leq p[l, \vec{\alpha}_l] - 1$. To do this, we must bound the minimum number of copies of $G(\vec{\gamma}')$ in $G\left(\vec{\gamma}', \vec{\alpha}_l, \vec{\beta}'\right)$ that contain pebbles (number of markers), during the traversal of this super edge. By definition,

$$p[l, \vec{\alpha}_l] \quad = \quad \max_{T \in T'[l, \vec{\alpha}_l]} \left[p + 1 - (\#\text{ copies of } G(\vec{\gamma}', \vec{\alpha}_l) \text{ containing pebbles at time } T)\right].$$

Therefore,

$$\min_{T \in T'[l-1, \vec{\alpha}_l]} [\#\text{ copies of } G(\vec{\gamma}', \vec{\alpha}_l) \text{ containing pebbles at time } T] \;\geq\; p - p[l, \vec{\alpha}_l] + 1.$$

We know that during the time interval $T'[l-1, \vec{\alpha}_l]$, one of the copies of $G(\vec{\gamma}', \vec{\alpha}_l)$ contains two copies of $G(\vec{\gamma}')$ (super edges) that contain pebbles. Therefore,

$$\min_{T \in T'[l-1, \vec{\alpha}_l]} [\#\text{ copies of } G(\vec{\gamma}') \text{ containing pebbles at time } T] \;\geq\; p - p[l, \vec{\alpha}_l] + 1 + \mathbf{1}$$

and, therefore, $p[l-1, \vec{\alpha}_l] \leq p[l, \vec{\alpha}_l] - 1$. From this we can bound the time of this super edge's traversal to be $T[l-1, \vec{\alpha}_l] \geq w[l-1, \vec{\alpha}_l] h^{-p\left[l, \vec{\alpha}_l\right]+1}$. Because the first event occurred, this occurred twice during disjoint

intervals in time. Hence, the time required for the first event to occur can be taken to be at least the sum of the times for the two super edges to be traversed, without over counting time steps. Therefore, $2 \times w[l-1,\vec{\alpha}_l]h^{-p\left[l,\vec{\alpha}_l\right]+1}$ time steps are required.

Case 2: Suppose the second event occurs and $\bigoplus_{j\in E_i}\left[\alpha_{\langle l,i\rangle}\right]_j = 1$. This event involves traversing up and down the handle. The handle contains $\frac{h}{2}$ super edges. Therefore, at least h super edges are traversed. These traversals must occur during disjoint intervals of time, because a super edge along the handle must be completely traversed, before the next super edge along the handle is entered. The maximum of number of pebbles $p[l-1,\vec{\alpha}_l]$ that can be used to traverse each of these copies of $G(\vec{\gamma}')$ is $p[l,\vec{\alpha}_l]$. Even if this number is used, the traversal time for each is $T[l-1,\vec{\alpha}_l] \geq w[l-1,\vec{\alpha}_l]h^{-p\left[l,\vec{\alpha}_l\right]}$. Therefore, the time to traverse h super edges is at least $w[l-1,\vec{\alpha}_l]h^{-p\left[l,\vec{\alpha}_l\right]+1}$.

Case 3: Suppose the second event occurs and $\bigoplus_{j\in E_i}\left[\alpha_{\langle l,i\rangle}\right]_j = 0$. Without loss of generality, assume that the pebbles entered the i^{th} fly swatter from the $(i-1)^{st}$ fly swatter through the $s_{\langle l,i\rangle}$ distinguished node. The pebbles then traverse up the handle through a sequence of switches specified by E_i and back down the handle to a distinguished node. Because $\bigoplus_{j\in E_i}\left[\alpha_{\langle l,i\rangle}\right]_j = 0$, the pebbles do not make it to the distinguished node $t_{\langle l,i\rangle}$, but arrive back at the $s_{\langle l,i\rangle}$ node. How do we know that the pebbles traverse back up and down the handle a second time? By the definition of $i \notin S_1$, the JAG must "continue on" to traverse into the $i+1^{st}$ fly swatter. Hence, they must traverse up and down the handles a second time. The two traversals up and down the handle take at least $2 \times w[l-1,\vec{\alpha}_l]h^{p\left[l,\vec{\alpha}_l\right]+1}$ time steps. ■

Using this claim, we can bound the total cost $w[l,\vec{\alpha}_l]$ (and time $T[l,\vec{\alpha}_l]$) for the JAG to traverse a line of fly swatters $G(\vec{\gamma}',\vec{\alpha}_l)$.

$$\begin{aligned} w[l,\vec{\alpha}_l] &= h^{p\left[l,\vec{\alpha}_l\right]}T[l,\vec{\alpha}_l] \\ &\geq h^{p\left[l,\vec{\alpha}_l\right]} \times \left[\sum_{i\in[1..d]-(S_0\cup S_1)} \left\{ \begin{array}{ll} 1 & \text{if } \bigoplus_{j\in E_i}[\alpha_i]_j = 1 \\ 2 & \text{otherwise} \end{array} \right\} w[l-1,\vec{\alpha}_l]h^{-p\left[l,\vec{\alpha}_l\right]+1}\right]. \end{aligned}$$

This can now be compared with the cost to the parity game defined above. By definition of the game, the number of questions asked per game instance on input $\vec{\alpha}_l$ is

$$\begin{aligned} c_{\vec{\alpha}_l} &= \frac{1}{d}\sum_{i\in[1..d]} \left\{ \begin{array}{ll} 1 & \text{if } \bigoplus_{j\in E_i}[\alpha_i]_j = 1 \\ 2 & \text{otherwise} \end{array} \right\} \\ &\leq \frac{1}{d}\left[2|S_0\cup S_1| + \sum_{i\in[1..d]-(S_0\cup S_1)} \left\{ \begin{array}{ll} 1 & \text{if } \bigoplus_{j\in E_i}[\alpha_i]_j = 1 \\ 2 & \text{otherwise} \end{array} \right\}\right]. \end{aligned}$$

By Claim 1, $|S_0 \cup S_1| \leq 2p+1$. This gives that

$$w[l,\vec{\alpha}_l] \geq w[l-1,\vec{\alpha}_l] \times h \times (dc_{\vec{\alpha}_l} - 4p - 2)$$

If $\vec{\alpha}_l$ were such that $w[l-1,\vec{\alpha}_l] \geq W[l-1]$ and $c_{\vec{\alpha}_l} \geq 1.5-\epsilon$, then

$$w[l,\vec{\alpha}_l] \geq W[l-1] \times h \times (d(1.5-\epsilon) - 4p - 2) = W[l]$$

It follows that

$$\Pr_{\vec{\alpha}_l}\left[w[l,\vec{\alpha}_l] < W[l]\right] \leq \Pr_{\vec{\alpha}_l}\left[c_{\vec{\alpha}_l} < 1.5-\epsilon\right] + \Pr_{\vec{\alpha}_l}\left[w[l-1,\vec{\alpha}_l] < W[l-1]\right]. \blacksquare$$

11 Completing the Proof

The final step is to prove the theorem.

Theorem 1′ *For every constant $z \geq 2$, and every helper-JAG algorithm for st-traversal that uses $p \leq \frac{1}{29z}\frac{\log n}{\log\log n}$ pebbles and $q \leq 2^{\log^z n}$ states and whose traversal time on input $G(\vec{\alpha}_1, ..., \vec{\alpha}_L)$ is $T_{\langle \vec{\alpha}_1,...,\vec{\alpha}_L\rangle}$,*

$$\Pr_{\langle \vec{\alpha}_1,...,\vec{\alpha}_L\rangle}\left[T_{\langle \vec{\alpha}_1,...,\vec{\alpha}_L\rangle} < n \times 2^{\frac{1}{29z}\frac{\log n}{\log\log n}}\right] \leq 2 \times 2^{-0.05\log^z n}.$$

Proof of Theorem 1′: Fix any constant $z \geq 2$ and consider a helper-JAG algorithm that uses $p \leq \frac{1}{29z}\frac{\log n}{\log\log n}$ pebbles and $q \leq 2^{\log^z n}$ states. The number of bits of help given by the helper JAG to set the initial configuration is the space of the JAG which is $b = p\log n + \log q \leq \frac{1}{29z}\frac{\log^2 n}{\log\log n} + \log^z n = (1+o(1))\log^z n$.

Consider the parity game with the bits of help from the helper being $b = (1+o(1))\log^z n$, the length of the vector $\vec{\alpha}_i$ being $r = 8$, the number of game instances being $d = 60\log^z n$, and the error tolerance being $\epsilon = 0.1$. By Lemma 1,

$$\begin{aligned}\Pr_{\vec{\alpha}}[c_{\vec{\alpha}} < 1.5-\epsilon] &\leq 2^b \times e^{-(2\epsilon^2 - 2^{-r+1})d} \\ &= 2^{(1+o(1))\log^z n} \times e^{-(2(.1)^2 - 2^{-8+1})60\log^z n} \\ &\leq 2^{-0.054\log^z n}\end{aligned}$$

Define the line of fly swatter graphs such that the number of switches per fly swatter graph is $r = 8$, the length of the handle is $h = \frac{10r}{0.05} = 1600$, and number of fly swatters in the line is $d = 60\log^z n$. Then, the number of vertices in $G(\vec{\alpha}_1, ..., \vec{\alpha}_L)$ is $n \leq [(h+10r)d]^L = [1.05hd]^L$ and $L \geq \frac{\log n}{\log(1.05hd)} \geq \frac{\log n}{z\log\log n + O(1)}$. Corollary 2 bounds the time $T_{\langle \vec{\alpha}_1,...,\vec{\alpha}_L\rangle}$ for the helper-JAG to traverse input $G(\vec{\alpha}_1, ..., \vec{\alpha}_L)$.

$$\begin{aligned}&\Pr_{\langle \vec{\alpha}_1,...,\vec{\alpha}_L\rangle}\left[T_{\langle \vec{\alpha}_1,...,\vec{\alpha}_L\rangle} < [h\times(d(1.5-\epsilon)-4p-2)]^L h^{-p+1}\right] \\ &\leq L \times \Pr_{\vec{\alpha}}[c_{\vec{\alpha}} < 1.5-\epsilon] \\ &\leq \frac{\log n}{z\log\log n + O(1)} \times 2^{-0.054\log^z n} \leq 2\times 2^{-0.05\log^z n}.\end{aligned}$$

This bound on the time can be computed to be

$$\begin{aligned}&[h\times(d(1.5-\epsilon)-4p-2)]^L \times h^{-p+1} \\ &\geq \frac{n}{[1.05hd]^L} \times [(1.4-o(1))hd]^L \times h^{-p+1} \\ &\geq n \times \left[\frac{1.4-o(1)}{1.05}\right]^L \times h^{-p} \\ &\geq n \times 2^{0.415\frac{\log n}{z\log\log n + O(1)}} \times 2^{-\log(1600)\times\frac{1}{29z}\frac{\log n}{\log\log n}} \\ &\geq n \times 2^{\frac{1}{29z}\frac{\log n}{\log\log n}} \quad \blacksquare\end{aligned}$$

The proof of Theorem 1 follows from Theorem 1′ using Yao [25] and Lemma 2.

12 The Pseudo Random Walk Game

The *st*-connectivity lower bound in Theorem 1 on the recursive fly swatter graphs reveals to the JAG every time in reaches the bottom of the handle whether it has reached the node $s_{\langle l,i\rangle}$ or the node $t_{\langle l,i\rangle}$. In reality,

however, the JAG does not have access to this information. My belief in fact is that the JAG quickly looses track of where the pebble is located in the line of fly swatter graphs. In effect, I believe that the pebble performs a "pseudo random walk" on the line. In this section, I define a game that characterizes this idea. A lower bound on this game would give the desired lower bound for st-connectivity of $n^{2-o(1)}$. The game is defined as follows.

Game 1 *The parameters of the game are d, r, and $\tilde{W}$ (ideally r is a constant, but it could be as much as $\log d$ and ideally $\tilde{W} = d^{2-\epsilon}$ but it could be as little as $2d$). The game consists of a player and a randomly chosen input, both of which direct a pebble walking pseudo randomly on a line of length d. First, the player outputs a fixed sequence of vectors $\vec{\gamma} = \langle \gamma_1, \gamma_2, \ldots \rangle \in [\{0,1\}^r]^*$. The vector γ_t is used at time t. Then the input is chosen. It is a sequence of vectors $\vec{\alpha} = \langle \alpha_1, \alpha_2, \ldots, \alpha_d \rangle \in (\{0,1\}^r - \{0^r\})^d$ that is chosen uniformly at random. The vector α_i is associated with the i^{th} node in the path.*

At every time step t, the pebble is located at some node i in the line and has a direction of travel from $\{\rightarrow, \leftarrow\}$. Initially, the pebble is at node 0 with direction $\rightarrow$. The transition depends on $\alpha_i \cdot \gamma_t = \bigoplus_{j \in [1 \ldots r]} [\alpha_i]_j \wedge [\gamma_t]_j$. If this dot product is 1 then the pebble takes a step in the direction that it is traveling in. If the dot product is 0 then the pebble turns around before taking a step. The following table lists the possibilities.

	$\alpha_i \cdot \gamma_t = 1$	$\alpha_i \cdot \gamma_t = 0$
$\overrightarrow{i}$	$i \overrightarrow{+} 1$	$i \overleftarrow{-} 1$
$\overleftarrow{i}$	$i \overleftarrow{-} 1$	$i \overrightarrow{+} 1$

Consider a fixed player specification $\vec{\gamma}$. If the pebble ever returns to node 0 then the pebble stops and the player looses. Otherwise, let $W_{\vec{\alpha}}$ be the number of time steps for the pebble to reach node d when the input is $\vec{\alpha}$. The goal of the player is to move the pebble quickly from node 0 to node d. Our goal is to prove that no matter what the player does, for a random input, the pebble will almost always require at least $\tilde{W}$ step, i.e. we win if

$$\forall \vec{\gamma} \ \Pr_{\vec{\alpha}}(W_{\vec{\alpha}} < \tilde{W}) \leq 2^{-d^{\epsilon}}.$$

The fact that the player loses if the pebble ever returns to node 0 is not in itself significant. The game could equivalently be defined so that the pebble bounces at node 0 back to node 1 or that the line is extended in both direction. What is significant about this part of the definition is that the player does not interact at all with the game. He outputs the vectors $\vec{\gamma} = \langle \gamma_1, \gamma_2, \ldots \rangle$ at the beginning of the game without any information about the input $\vec{\alpha}$.

I think that a constant $r \in O(1)$ random bits per node is sufficient. In Theorem 1, it is 8. This is large enough so that $\{0,1\}^r - \{0^r\}$ is not significantly far from $\{0,1\}^r$. On the other hand, maybe more random bits may help us. However, we have reasons to believe that having $r > \log d$ can only help the player.

Lemma 4 *For $r \in O(1)$ and $\tilde{W} = d^{2-\epsilon}$, the player can win.*

Proof of Lemma 4: For a purely random walk on a line, $\Pr_{\vec{\alpha}}(W_{\vec{\alpha}} < d^{2-\epsilon}) \leq 2^{-d^{\epsilon}}$. The player can easily be given to match this bound by outputting a random sequence of $\vec{\gamma}$. ■

My conjecture is that the player in this random-walk game cannot do significantly better than by outputting a random sequence of $\vec{\gamma}$.

Conjecture 1 *For some $r \in [8 \ldots \log d]$ and for some $\tilde{W} \in [d^{1+\epsilon} \ldots d^{2-\epsilon}]$, the player must lose.*

Lemma 1 effectively proves this conjecture for $r = 8$ and $\tilde{W} = 2d$. The following theorem states the JAG result.

Theorem 3 *Suppose that Conjecture 1 is true, i.e. the pebble in Game 1 almost always takes d^ω time steps to traverse a line of length d, for some $\omega \in [1+\epsilon ... 2-\epsilon]$. It follows that for every constant $z > 2$, the expected time to solve undirected st-connectivity on a JAG with $p \leq \frac{\delta^2 \log n}{2z \log\log n}$ pebbles and $q \leq 2^{\log^z n}$ states is at least $n^{\omega(1-2\delta)}$.*

The proof reduces *st*-traversal of recursive fly swatter graphs by a helper-JAG to the pseudo random walk game. This reduction requires quite a different proof than the one presented here. It has not been included because it is slightly more difficult and because a lower bound on the pseudo random walk game has not been obtained.

References

[1] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovász, and C. 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, October 1979. IEEE.

[2] Greg Barnes, Uriel 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.

[3] Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch 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.

[4] Greg Barnes and Jeff Edmonds. Time-space lower bounds for directed *st*-connectivity on JAG models. In *34th Annual Symposium on Foundations of Computer Science*, pages 228–237, Palo Alto, CA, November 1993.

[5] P. Beame, A. Borodin, P. Raghavan, W. L. Ruzzo, and M. Tompa. Time-space tradeoffs for undirected graph connectivity. In *31st Annual Symposium on Foundations of Computer Science*, pages 429–438, St. Louis, MO, October 1990. IEEE. Full version submitted for journal publication.

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

[7] 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, October 1992.

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

[9] A. Z. Broder, A. R. Karlin, P. Raghavan, and E. Upfal. Trading space for time in undirected *s-t* connectivity. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 543–549, Seattle, WA, May 1989.

[10] 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. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 574–586, Seattle, WA, May 1989.

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

[12] Jeff Edmonds. *Time-Space Lower Bounds for Undirected and Directed st-Connectivity on JAG Models.* PhD thesis, Department of Computer Science, University of Toronto, August 1993.

[13] Jeff 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.

[14] Russell Impagliazzo and David Zuckerman. How to recycle random bits. In *30th Annual Symposium on Foundations of Computer Science*, pages 248–253, Research Triangle Park, NC, October 1989. IEEE.

[15] Jeff D. Kahn, Nathan Linial, Noam Nisan, and Michael E. Saks. On the cover time of random walks on graphs. *Journal of Theoretical Probability*, 2(1):121–128, January 1989.

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

[17] Noam Nisan, Endre Szemerédi, and Avi Wigderson. Undirected connectivity in $O(\log^{1.5} n)$ space. In *33rd Annual Symposium on Foundations of Computer Science*, Pittsburgh, PA, October 1992. IEEE.

[18] C. K. Poon. A sublinear space, polynomial time algorithm for directed *st*-connectivity on the JAG model. Manuscript.

[19] Steven Rudich. personal communication.

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

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

[22] N. A. Spencer. John Wiley and Sons, Inc., 1992.

[23] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, June 1972.

[24] A. Wigderson. The complexity of graph connectivity.

[25] A. C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *18th Annual Symposium on Foundations of Computer Science*, pages 222–227, Providence, RI, October 1977. IEEE.