Time-Space Lower Bounds for Undirected and Directed ST-Connectivity on JAG Models

by

Jeff A. Edmonds

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract – Time-space lower bounds for undirected and directed st-connectivity on JAG models

Jeff A. Edmonds
Doctor of Philosophy 1993
Department of Computer Science
University of Toronto

Directed and undirected st-connectivity are important problems in computing. There are algorithms for the undirected case that use $O(n)$ time and algorithms that use $O(\log n)$ space. The first result of this thesis proves that, in a very natural structured model, the JAG (Jumping Automata for Graphs), these upper bounds are not simultaneously achievable. This uses new entropy techniques to prove tight bounds on a game involving a helper and a player that models a computation having precomputed information about the input stored in its bounded space. The second result proves that a JAG requires a time-space tradeoff of $T \times S^{\frac{1}{2}} \in \Omega\left(mn^{\frac{1}{2}}\right)$ to compute directed st-connectivity. The third result proves a time-space tradeoff of $T \times S^{\frac{1}{3}} \in \Omega\left(m^{\frac{2}{3}}n^{\frac{2}{3}}\right)$ on a version of the JAG model that is considerably more powerful and general than the usual JAG.

Acknowledgments

There are many people I wish to thank for helping me along the path from s to t. There were those who held my hand while I walked the treacherous edges from idea to idea and bug to bug. I am also indebted to those who helped me make sure that my ideas changed state from thought to written form in a way that insured they were accepted rather than rejected. As I jumped from deadline to deadline, there were always people who insured that I cleared the hurdles and who gave me the support I needed to prepare for the next leap. Faith Fich, my advisor, provided the perfect balance between directing my path and leaving me to take a random walk through the undirected graph of research.

Courses and research groups were important in providing the necessary supplies for my trip. They helped me learn about new papers, hear about the research of my peers, and gave me opportunities to share my own work. Some of my favorite courses were combinatorial methods by Faith and Mauricio Karchmer, logic and non-standard numbers by Russell Impagliazzo and Steve Cook, graph theory by Derek Corneil, distributed systems by Vassos Hadzilacos, and complexity by Al Borodin. The research groups that most influenced me were graph theory led by Derek Corneil, NC_2 led by Mauricio Karchmer, ordinals and graph minors led by Arvind Gupta, the probabilistic method led by Hisao Tamaki, and complexity theory led by Al Borodin. The breadth requirements were important as well. Some of them were actually interesting and taking them with Gara Pruesse made them all the more enjoyable. Especially, the beer afterwards.

When I was traversing some of those edges during my mathematical odyssey, there were many people who provided valuable support. Russell Impagliazzo, Arvind Gupta, Toni Pitassi, Hisao Tamaki, Tino Tamon, and Chung Keung Poon were kind enough to listen to my ideas and help me along. Some of my best work was also done jointly with these same people. Hisao, Tino, C.K. and Russell were particularly helpful in providing me with information from the mathematical literature that I wasn't as well versed in as I should have been. Al Borodin, Charlie Rackoff and Steve Cook helped me by hitting my bugs with a fly swatter. And of course, a great deal of thanks goes to Faith Fich for treating me as a peer and for always being excited to discuss research.

I will always be grateful to those who helped me convert my research to understandable English. Faith Fich was my most devoted critic and English teacher and spent much of her time beating my writing into shape and combing out the typos. Since criticism is often hard to hear, I sometimes relied on Jeannine St. Jacques to help put it into perspective. Many proof readers were also crucial in revising my work. In addition to Faith Fich, I would like to thank Toni Pitassi, Naomi Nishi, Jennifer Edmonds, Miriam Zachariah, Paul Beame, and some unknown referees who were very hard working. I was also saved many times by Steven Bellantoni, Eric Schenk, and Ken Lalonde when I was lost with Unix and Latex.

There were many people who provided the emotional support I needed to continue my journey.

Faith Fich, Russell Impagliazzo, and Arvind Gupta, who had already become successful researchers, often reassured me when I despaired about my own success. Faith was always there for me with motherly love. Through out the years, Al too gave many touching words of support that were greatly appreciated. I thank Derek Corneil, Arvind Gupta, Faith Fich, Steven Rudich, and Pat Dymond for arranging for me to give talks outside of U. of T. which provided me with some of the exposure and experience I needed. Martha Hendricks lightened things up and provided distractions when I was working too hard. Rick Wallace, Shirley Russ, Roland Mak, and Toni Pitassi were always by my side to help me with ongoing emotional issues.

PhD students are responsible for more than their own research. As I learned to teach undergraduates, I found important role models in Jim M^{c}Innes, Peter Gibbons, and Faith Fich. Kathy Yen, Teresa Miao, and Vicky Shum led me by the hand when I had administrative obligations.

I would be remiss if I did not mention a few others who helped me on my way. My father, Jack Edmonds, deserves some of the credit for my introduction to mathematics. He continually kept me interested in puzzles to ponder. My mother, Patricia Oertel, provided support and encouragement as I made my way through public school and university. Baby Joshua was very obliging in keeping me awake at night so that I could do research. He also provided a lots of joy and love. My wife Miriam was a tremendous support in every way possible and one last thanks goes to Faith Fich for being such a wonderful advisor.

Contents

Chapter 1

Introduction

This thesis proves lower bounds on the time-space tradeoffs for computing undirected and directed *st*-connectivity on the JAG and related models of computation. This introductory chapter introduces the models of computation considered, gives a brief history of time-space tradeoffs, summarizes the known upper and lower bounds for *st*-connectivity, and outlines the contributions of the thesis.

1.1 JAGS, NNJAGs, and Branching Programs

Many different models of computation are used in computer science. These abstract the crucial features of a machine and allow theoreticians, programmers, and architecture designers to communicate in succinct ways. Another motivation for defining models is to be able to prove lower bounds. A long term goal is to be able to prove lower bounds on the *Random Access Machines* (RAM), because this is the abstraction that best models the computers used in practice. However, this goal remains elusive.

In order to make the task of proving lower bounds easier, models are defined that are in some ways more restricted than the RAM and in some ways are less restricted. One way of restricting the model is to restrict the order in which it is allowed to access the input. Another way is to not allow the actions of the model to depend on the aspects of the input that have to do with the input representation and not on the output of the computational task at hand. The usual justification for restricting the model in these ways is that it seems natural and that most known algorithms for the given problem adhere to the restrictions.

One way to strengthen a model is to define the model's internal workspace in such a way that no assumptions are made about how the space is managed. Another way to strengthen the model is to add to its instruction set; in the extreme, allow the model to compute any function of the known information in one time step.

This thesis is particularly interested in two models; JAGs and NNJAGs, and is indirectly interested in branching programs. These models and some closely related models are defined below. In Figure 1.1, the relative strengths in terms of direct simulations between these models, the RAM and the Turing machine are depicted.

The **(r-way) branching program** [BC82] is the most general and unstructured model

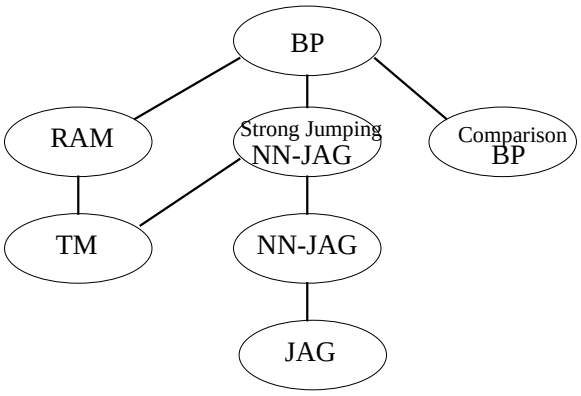

Figure 1.1: Models of Computation

of sequential computation and is at least as powerful as all reasonable models of computation. Depending on the current state, one of the input variables is selected and its value is queried. Based on this value, the state changes. These state transitions are represented by a directed acyclic rooted graph with out-degree r, where r is the maximum number of different values possible for an input variable. Each node of this graph represents a possible state that the model might be in and is labeled with the input variable that is queried in this state. The edges emanating out of the node are labeled with the possible values of the queried variable and the adjacent nodes indicate the results of the corresponding state transitions. In order to indicate the outcome of the computation, each sink node is labeled with either *accept* or *reject*. A computation for an input consists of starting at the root of the branching program and following a **computation path** through the program as explained above, until an *accept* or *reject* sink node is reached. The time T_G is the length of the computation path for input G. The space S is formally the logarithm of the number of nodes in the branching program. Equivalently, it can be viewed as the number of bits of workspace required to specify the current state. The branching program is said to be **leveled** if the nodes can be assigned levels so that the root has level 0 and all edges go from one level to the next.

The branching program model is more powerful than the RAM model in the two ways mentioned above. Assumptions are made about how the RAM manages its workspace. Specifically, the workspace is broken into many small squares and the RAM is only allowed to alter one of these squares at a time. What may be more significant is that, during a particular time step, the RAM's next action is allowed to depend only on a finite state control together with the contents of only a few of these input squares. In contrast, the state of the branching program specifies the contents of the entire workspace. A branching program is able to alter the entire workspace in one time step by means of the appropriate state transition and its next action can depend on its new state in an arbitrary way. Because any transition is allowed, no assumption is made about the way the model's workspace is managed.

A RAM is also restricted in comparison to a branching program because it has a limited instruction set. A RAM can compute the sum or the product of the contents of two squares in one step, but is not, for example, able to quickly factor these numbers. A branching program does not have this restriction. Its state transition function can be defined arbitrarily and non-uniformly. Hence, in one time step, it can compute any function of the known information in a manner that amounts to a table lookup.

Although in many ways unrealistic, the advantage of proving lower bounds on the more

powerful branching program model is that we gain the understanding of what features of the computational problem really are responsible for its difficulty. Another benefit is that a number of proof techniques have been developed for the branching program model, whereas we have few tools to take advantage of the fact that RAM has a restricted instruction set or that it can change only one bit of its workspace at a time. Besides, any lower bounds for the branching program apply to the RAM.

Proving lower bounds for decision problems on either the branching program model or the RAM is beyond the reach of current techniques. Hence, restricted models of computation are considered. One model that has been a successful tool for understanding the complexity of graph connectivity is the "jumping automaton for graphs" (JAG) model introduced by Cook and Rackoff [CR80]. This model restricts the manner in which it is allowed to access the input and in the type of information about the input that it is allowed to access. In addition, some of its workspace is structured to contain only a certain type of information. As well, its basic operations are based on the structure of the graph, as opposed to being based on the bits in the graph's encoding [Bo82]. Hence, it is referred to as a "structured" model of computation. The JAG model has a set of pebbles representing node names that a structured algorithm might record in its workspace. They are useful for marking certain nodes temporarily, so that it can be recognized when other pebbles reach them.

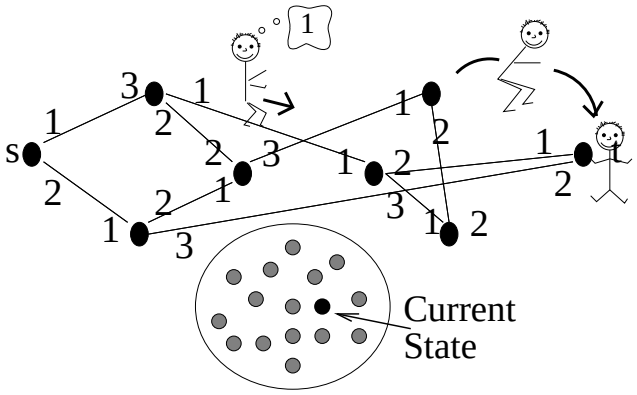

Figure 1.2: The JAG model for undirected graphs

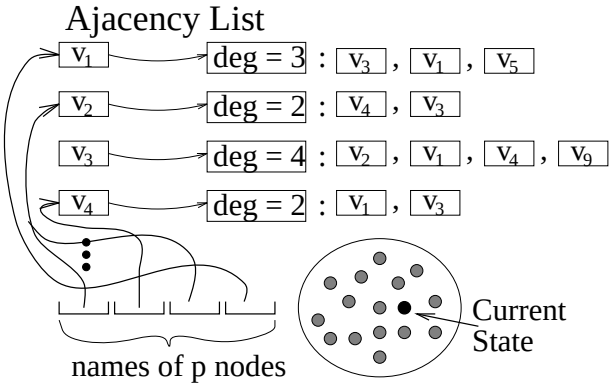

Figure 1.3: The structured allocation of the workspace that the JAG models

The **JAG** [CR80] is a finite automaton with p distinguishable pebbles and q states. The

space charged to the model is defined to be $S = p \log_2 n + \log_2 q$. This is because it requires $\log_2 n$ bits to store which of the n nodes a pebble is on and $\log_2 q$ bits to record the current state.

The input to this JAG model is a graph with n nodes, m edges, and two distinguished nodes s and t. The input graph is directed or undirected depending on whether directed or undirected st-connectivity is being considered. For each node v, there is a labeling of the out-going edges with the numbers from 1 to the out-degree of the node. (If the edge is undirected, it can receive two different labels at its two endpoints.) One of the pebbles is initially placed on the distinguished node t and the other $p - 1$ are placed on s. The initial state is Q_0.

Each time step, the automaton is allowed an arbitrary non-uniform state transition similar to that of a branching program. It also does one of the following two things. It either selects some pebble $P \in [1..p]$ and some label l and **walks** P along the edge with label l or it selects two pebbles $P, P' \in [1..p]$ and **jumps** P to the node occupied by P'. (For directed graphs, the pebbles may be walked in the direction of an edge, but not in the reverse direction.)

What the JAG chooses to do each time step is only allowed to depend on specific parts of the current machine configuration. Specifically, its actions are able to depend on the current state, which pebbles are on the same nodes, which pebbles are on the distinguished nodes s and t, and the out-degrees of the nodes containing pebbles[1]. (For directed graphs, the model has no direct access to the in-degrees of the nodes.)

A JAG that solves undirected st-connectivity enters an accepting state if there is an undirected path between s and t in the input graph and enters a rejecting state if there is no such path. Similarly, a JAG that solves directed st-connectivity enters an accepting state if and only if there is a directed path from s to t.

Now that it has been specified what the JAG model is allowed to do, I will make it clearer what it has not been allowed to do. It is, in fact, restricted in the two ways mentioned at the beginning of this section.

Like the Turing machine, the JAG model is restricted in the order in which it is allowed to access its input. The Turing machine is only allowed to move its input heads one square to the right or one to the left in a single step. The JAG is only allowed to "walk" and "jump" its pebbles. For the task of graph st-connectivity, accessing the input in an order constrained by the Turing machine's tape does not seem to be natural. In fact, such a restriction would likely increase the complexity of the problem by a factor of $O(n)$. [2] In contrast, "walking" pebbles along the edges of the input graph does seem to be natural to the computational problem at hand. After all, the model can know that s is connected to t simply by "walking" a pebble from s to t.

Being able to "jump" a pebble directly to the node containing another pebble also adds a great deal of power to the JAG model that the Turing machine does not have. From lower bounds on JAGs that are not allowed to jump (WAGs) [BRT92, BBRRT90], it can be concluded that jumping increases the model's power significantly, because the model is able to quickly concentrate its limited pebble resources in the subgraph it is currently working on.

At first, it might seem unnatural that the JAG model for directed graphs is not allowed to

[1] The graphs in [CR80] are d-regular for some fixed degree. I am allowing the input to be more general. The JAG is told the out-degree of the nodes containing pebbles so that it knows which labels l it can choose for walking a pebble. The distinction is of greater importance when the model is being charged for the computation time.

[2] [CR80] charged the model only for space. I am charging the model for time as well. Hence, these factors are important.

access the in-degree or allowed to walk in the reverse direction along an edge. However, suppose the input is stored by listing, for each node, the out-degree followed by the out-adjacency list. Then, even a RAM, would not be able to learn the in-adjacency list of a node without scanning the entire input[3]. Hence, not allowing the JAG to walk pebbles backwards along directed edges is reasonable and natural.

Even when the order in which the model is allowed to access its input is restricted, proving lower bounds for decision problems like st-connectivity is hard. One reason is that a general model of computation is able to perform weird tasks and store arbitrary information. Hence, it is difficult for us to judge, for each of the possible states that the model might be in, how much progress has been made towards solving the given problem. By restricting the model again so that it is only able to learn or store "structured" information about the input, it is then possible to define a measure of progress.

More precisely, any input representation of a graph that is written as a list of values must impose "names" on the nodes of the graph. With these names, a general model of computation is able to do strange things like taking their bitwise parity and then changing its actions based on the values obtained. In contrast, the JAG is not allowed to do things which seem to be unnatural. This restriction can viewed as a restriction on the instruction set of the JAG or, equivalently, as a change in the way that the input is represented. The input graph can be considered to be as an abstract object, in which the nodes other than s and t do not have names. The actions of the JAG do not depend on the names of these nodes. Effectively, this means that the actions of the JAG are not allowed to depend on which nodes the pebbles are on, but only on which pebbles are on the same nodes. Note that the answer to a problem instance and hence the final outcome of a computation does not depend on the names of the non-distinguished nodes.

The JAG is not the only model that is restricted in how it can react to its input. The comparison based branching program is another example. In this model, a step consists of comparing two inputs variables and changing state based on which is larger. Hence, the actions of a comparison based branching program are allowed to depend on the permutation describing the relative ordering of the input variables, but not on the actual values of these variables. Similarly, an arithmetic circuit model is able to add, subtract, and multiply the values of certain input variables, but the choice of operations and the variables cannot depend on the actual values that variables have.

Despite the fact that the JAG is restricted in two significant ways, the JAG is not a weak model of computation. To begin with, it is not strictly weaker than the RAM, because like a branching program, it allows arbitrary non-uniform state transitions. More importantly, it is general enough so that most known algorithms for graph connectivity can be implemented on it. See Section 1.3 for some upper bounds. Furthermore, Beame, Borodin, Raghavan, Ruzzo, and Tompa [BBRRT90] give a reduction that transforms any $O(s)$ space Turing machine algorithm for any undirected graph problem to run on a $O(s)$ space JAG within a polynomial factor as fast. Hence, a lower bound for the JAG translates to a lower bound for a Turing machine. However, this is not helpful here because $\Omega(n^2)$ time bounds for the JAG are too small as compared to the overhead incurred by this reduction.

There has been a great amount of success in proving lower bounds on the JAG model. Poon [Po93b] has taken this work another step further by introducing another model that is considerably

[3] An interesting related problem is the following. The input is a set of n pointers forming a set of linked lists of length l and the last node of one of the lists. The output is the head of the list whose last node has been specified. I conjecture that a RAM with $O(\log n)$ space would require $\Theta(ln)$ time.

less restricted. The model is referred to as the *node named jumping automaton for graphs* (NNJAG). The initial motivation for defining this model was to overcome an apparent weakness of the JAG: namely, on the JAG, it does not seem possible to implement Immerman's non-deterministic $O(\log n)$ space algorithm for detecting that a graph is not st-connected. However, Poon can do so on an NNJAG.

The **NNJAG** is an extension of the JAG, with the additional feature that it can execute different actions based on which nodes the pebbles are on. This feature makes the NNJAG a considerably more general model of computation than the JAG. The NNJAG is able to store arbitrary unstructured information about the names of the nodes by transferring the information that is stored in the "pebble workspace" into the "state workspace". This is done by having the state transition depend on the names of the nodes containing pebbles.

If the NNJAG was to be able to access its input in an arbitrary order, then the model would be equivalent to the branching program model. However, it is not. Although it can store any information in its state, it can only gain new information about the input graph by means of its pebbles. In addition, even if the model knows the name of a particular node, it is only allowed to "jump" a pebble there if another pebble is already there and to get a pebble there initially, it must "walk".

The NNJAG seems incomparable to the Turing machine, because they are allowed to access their inputs in different orders. Savitch introduced a new operation, **strong jumping**, to the JAG model [Sa73]. Strong jumping is the ability to move any pebble from a node to the next higher numbered node, for some ordering of the nodes. Such an operation seems to be comparable to jumping a pebble to an arbitrary node of the graph, because the nodes can be placed by an adversary in an arbitrary order. On the other hand, with this additional power, the NNJAG model is strictly more powerful than the Turing machine model by direct simulation. Because the NNJAG lacks this ability, it is not able scan to the entire input in linear time. In fact, pebbles are never able to reach components of the input graph that do not initially contain pebbles. For solving st-connectivity, it does not seem that the model could gain useful information by doing this, but this restriction does mean that the JAG and the NNJAG are not considered generals model of computation. At present, we are unable to remove this restriction on the model, because it is this feature that enables us to prove lower bounds for JAGs. Recall that a difficulty in proving lower bounds is defining a measure of the amount of progress that a computation has done. For the NNJAG model, this problem is solved by defining progress in terms of how much of the graph the model has accessed so far.

All the above mentioned models, the branching program, the JAG, and the NNJAG are deterministic. Probabilistic versions of these models can be defined quite simply. For every random string $R \in \{0,1\}^*$, the model will have a separate algorithm, with disjoint states. At the beginning of the computation a random $R \in \{0,1\}^*$ is chosen. Then the computation proceeds as before with the selected algorithm.

The space of a probabilistic JAG, NNJAG, or branching program is defined to be the maximum of the space used by each of the separate algorithms. This effectively provides the model with $|R|$ bits of read only workspace which contains R and which can be accessed in its entirety every time step.

As long as the model has enough space to store the time step, i.e. $S \geq \log T$, this way of representing probabilism is stronger than supplying the model with a few random bits each time

step[4]. In this case, the time step can be used as a pointer into R. Yao uses this same definition of randomization [Ya77]. However, for upper bounds this definition is unrealistic. A non-uniform algorithm for one value of n can be specified in polynomial size by giving the circuit. However, specifying such a random algorithm requires a non-uniform circuit for each random string.

A probabilistic algorithm is said to allow **zero-sided error** if the correct answer must always be produced; however, the running time may depend on the choice of the random string R. A probabilistic algorithm is said to allow **one-sided error** if for every input not in the language, the correct answer is given, but for inputs in the language the incorrect answer may be given with some bounded probability. A probabilistic algorithm is said to allow **two-sided error** if for every input, it may give the incorrect answer with probability $\left(\frac{1}{2} - \epsilon\right)$.

We have now defined models of computation that are both non-uniform and probabilistic. In many circumstances, this is redundant. Adleman tells us how to convert a probabilistic algorithm into a non-uniform deterministic algorithm [Ad78]. He does this by proving that, for every probabilistic algorithm, there exists a set of $O(n)$ random strings R, such that for every input, one of the random strings gives the correct answer. The non-uniform algorithm is to simply deterministically run the probabilistic algorithm for each of these choices of R. An effect of this reduction is that a lower bound on a non-uniform deterministic model applies for free to the probabilistic version of the model. The only difficulty is that, because the reduction blows up the time by a factor of $O(n)$, a lower bound of $\Omega(n^2)$ on the non-uniform deterministic model would say nothing non-trivial about the probabilistic model. Therefore, there is value to proving lower bounds on non-uniform probabilistic models of computation. Clearly, such a bound applies to uniform probabilistic models as well.

Yao compares worst case probabilistic algorithms and average case deterministic algorithms [Ya77]. He proves that there exists a probabilistic algorithm whose expected running time on the worst case input is T if for **every** distribution on inputs, there exists a deterministic algorithm whose average running time weighted by the input distribution is T. This means that if there is not a probabilistic algorithm, then there exists a distribution for which there is no algorithm that is fast on average with respect to this distribution. This thesis will strengthen the lower bound further by specifying a specific natural input distribution for which the average time must be large. In fact, this thesis considers both of these settings at the same time, i.e. average case probabilistic algorithms. Given Yao's result, this might be excessive, but it is helpful for forming additional intuition.

The lower bounds proved are on probabilistic JAGs for undirected graphs, JAGs for directed graphs, and probabilistic NNJAGs for directed graphs.

1.2 Time Space Tradeoffs

Proving time-space tradeoffs is one of the more classical issues in complexity theory. For some computational problems it is possible to obtain a whole spectrum of algorithms within which one can trade the time requirements with the storage requirements. In these cases, the most meaningful

[4]If the space is less than this, then the random bits cannot be chosen initially, because the model cannot remember them. Beame et al. [BBRRT90] proved that a deterministic WAG with $pq = \Omega(n)$ traversing 2-regular graphs have infinite worst case time. Hence, with the random bits provided at the beginning of the computation, the expected time would be infinite. However, a probabilistic WAG can easily traverse such a graph in expected n^2 time.

bounds say something about time and space simultaneously.

Cobham [Co66] established the first time-space tradeoff. The model used was a one tape Turing Machine. Tompa [Tm80] established a number of time-space tradeoffs for both the Boolean and the arithmetic circuit models.

Borodin, Fischer, Kirkpatrick, Lynch, and Tompa [BFKLT81] introduced a framework that has been extremely successful in proving lower bounds on the time-space tradeoffs for branching programs. Using this framework, they prove the near optimal bound of $T \times S \in \Omega\left(n^2\right)$ for sorting on comparison based branching programs. Borodin and Cook [BC82] strengthened this result significantly by proving the first non-trivial time-space tradeoff lower bound for a completely general and unrestricted model of computation, the r-way branching program. They showed that $T \times S \in \Omega\left(\frac{n^2}{\log n}\right)$ for sorting n integers in the range $[1..n^2]$. Beame [Be91] improved this to $\Omega\left(n^2\right)$. These same techniques were also used to prove lower bounds for a number of algebraic problems such as the FFT, matrix-vector product, and integer and matrix multiplication [Ye84, Ab86, BNS89].

Each of these lower bounds relies heavily on the fact that the computation requires many values to be output, in order to establish a measure of progress. For decision problems this method does not work. Borodin, Fich, Meyer auf der Heide, Upfal, and Wigderson [BFMUW87] define a new measure of progress and prove a lower bound on the time-space tradeoff for a decision problem. They proved $T \times S \in \Omega\left(n^{\frac{3}{2}} \log n\right)$ for determining element distinctness. This was then improved to $T \times S \in \Omega\left(n^{2-\epsilon}\right)$ by Yao [Ya88]. They were not, however, able to prove this bound for a general model of computation, but only for the comparison based branching program.

1.3 The *st*-Connectivity Problem

Graph connectivity is an important problem, both practically and theoretically. Practically, it is a basic subroutine for many graph theoretic computations. It is used in solving network flow optimization problems, such as project scheduling and the matching of people to jobs. Theorem proving can be viewed as finding a logical path from the assumptions to the conclusion. 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 undirected version was used by Karchmer and Wigderson to separate monotone NC_1 from NC_2. The importance of these problems is discussed in more detail in Wigderson's beautiful survey [Wi92].

The fastest algorithms for undirected graph *st*-connectivity are depth-first and breadth-first search [Tar72]. These use linear time, i.e. $O\left(m+n\right)$ for an n node, m edge graph. However, they require $\Omega\left(n\right)$ space on a RAM.

Alternatively, this problem can be solved probabilistically using random walks. The expected time to traverse any component of the graph is only $\Theta\left(mn\right)$ and uses only $O\left(\log n\right)$ space [AKLLR79]. More generally, Broder et al. [BKRU89] have exhibited a family of probabilistic algorithms that achieves a tradeoff of $S \cdot T \in m^2 \log^{O(1)} n$ between space and time. This has been improved to $S \cdot T \in m^{1.5} n^{.5} \log^{O(1)} n$ [BF93]. A long term goal is to prove a matching lower bound.

Deterministic non-uniform algorithms for *st*-connectivity can be constructed using *Universal traversal sequence*. The algorithm uses a single "pebble" to traverse the connected components of the graph. Sequences of length $O(n^4 \log n)$ are proved to exist [CRRST89, KLNS89]. This proves the existence of $O(\log n)$ space and $O(n^4 \log n)$ time algorithms. Nisan provides an explicit constructions of a universal traversal sequences using pseudo-random generators. This gives a deterministic uniform algorithm using $O\left(\log^2 n\right)$ space and $n^{O(1)}$ time [Ni92]. Finally, Nisan, Szemerédi, and Wigderson describe and a deterministic uniform algorithm that uses only $O\left(\log^{1.5} n\right)$ space [NSW92]. The time for this algorithm, however, is $2^{O\left(\log^{1.5} n\right)}$.

Directed *st*-connectivity has the same complexity as undirected *st*-connectivity when there is ample space. Specifically, the linear time, linear space, depth first search algorithm works fine. However, the known algorithms for the directed problem require considerably more time when the space allocated to the model is bounded. Savitch's algorithm [Sa70] uses only $O\left(\log^2 n\right)$ space, but requires $2^{O(\log^2 n)}$ time. It is only recently that a polynomial time, sub-linear $O\left(\frac{n}{2^{\sqrt{\log n}}}\right)$ space algorithm was found for directed *st*-connectivity [BBRS92].

Section 1.1 stated that the JAG model is general enough so that most known algorithms for graph connectivity can be implemented on it. I will now be more specific. It 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. As well, because it is a non-uniform model, the JAG is able to solve *st*-connectivity for undirected graphs in $O(\log n)$ space and $O(n^4 \log n)$ time using a universal traversal sequence [CRRST89, KLNS89]. For directed graphs, Cook and Rackoff [CR80] show that the JAG model is powerful enough to execute an adaptation of Savitch's algorithm [Sa70] which uses $O\left(\log^2 n\right)$ space. Poon [Po93a] shows that Barnes et al.'s [BBRS92] sub-linear space, polynomial time algorithm for directed *st*-connectivity runs on a JAG as well.

1.4 A History of Lower Bounds for *st*-Connectivity

A number of space lower bounds have been obtained (even when an unbounded amount of time is allowed). Cook and Rackoff [CR80] 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 [BS83] (for $T \in 2^{\log^{O(1)} n}$). Recently, Poon [Po93b] has extended this result again to the powerful probabilistic NNJAG model.

For undirected graph *st*-connectivity, Cook and Rackoff [CR80] prove that $pq \in \omega(1)$ and Beame et al. [BBRRT90] 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 at least a linear number of states, because then the JAG is able to count the time steps.)

Lower bounds on the tradeoff 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. For example, a *universal traversal sequence* is simply a JAG with an unlimited number of states, but only one pebble. Borodin, Ruzzo, and Tompa [BRT92] prove that on this model, undirected *st*-connectivity requires $\Omega(m^2)$ time where the graph has degree $3 \leq d \leq \frac{1}{3}n - 2$. Beame, Borodin, Raghavan, Ruzzo, and Tompa

[BBRRT90] extend this to $\Omega\left(n^2/p\right)$ time for p pebbles on 3-regular graphs with the restriction that all but one pebble are unmovable (after being initially placed throughout the graph). Thus, for this very weak version of the model, a quadratic lower bound on time×space has been achieved. Beame et al. [BBRRT90] 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.

1.5 The Contributions of the Thesis

This thesis proves time-space tradeoffs for both undirected and directed st-connectivity. The first result, presented in Chapter 3 proves the strongest known bound for undirected st-connectivity. The result is that the expected time to solve undirected st-connectivity on a JAG is at least $n \times 2^{\Omega\left(\frac{\log n}{\log\log n}\right)}$, as long as the number of pebbles $p \in O\left(\frac{\log n}{\log\log n}\right)$ and the number of states $q \in 2^{\log^{O(1)} n}$. This result improves upon at least one of 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 only three, it applies to an **average case** input instead of just the worst case input, and **probabilistic** algorithms are allowed.

This result is obtained by reducing the st-connectivity problem to the problem of traversing from s to t and then reducing this problem to a two player game. Chapter 2 proves tight bounds for this game. The game is referred to as the *helper-parity game* and is designed to mirror a key aspect of time-space tradeoffs. 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. The difficulty in proving lower bounds is that we must assume that earlier stages of the computation communicate partial information about the subproblem via the bounded workspace. The question then is how helpful this information can be in decreasing the time to recompute the subproblem by the later stages. This is modeled in the helper-parity game by having the helper communicate the stored information to the player at the beginning of the player's computation. Upper and lower bounds are provided giving the tradeoff between the amount of information that the helper provides and the time for the player's computation.

Chapter 4 proves the even stronger lower bound of $T \times S^{\frac{1}{2}} \in \Omega\left(mn^{\frac{1}{2}}\right)$, but for the more difficult computational problem of directed st-connectivity. This is the first time-space tradeoff where the pebbles are able to jump and their number is unrestricted. Note, as well, that the time bound matches the upper bound of $O(m+n)$ when the amount of available space increases to n. Another interesting feature of the result is that it does not count the number of states as part of the space and hence applies even when the JAG has an arbitrarily large number of states. It had been assumed that the time to compute st-connectivity on a JAG became linear as the number of states increases.

Chapter 5 then proves a weaker bound of $T \times S^{\frac{1}{3}} \in \Omega\left(m^{\frac{2}{3}} n^{\frac{2}{3}}\right)$ on the same family of graphs, but on the more powerful probabilistic NNJAG model. Very different techniques are required to accomplish this. The general framework is the one outlined in Section 1.2.

The last chapter describes some current work and open problems.

Chapter 2

The Helper-Parity Game

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. The difficulty in proving lower bounds is that only in the first encounter with a subproblem is the computation completely without knowledge about the solution. We must assume that the earlier stages of the computation communicate partial information about the subproblem via the bound workspace. This chapter characterizes this problem in terms of a game referred to the *helper-parity game*. The helper in the game communicates to the player the information that would be stored in the bounded workspace. The player then uses this information to help in his computation. Upper and lower bounds are provided in this chapter, giving the tradeoff between the amount of information that the helper provides and the time for the player's computation.

The hope was that requiring the player to solve multiple instances of the problem would make proving time-space tradeoffs easier. Because the space is bounded, the number of bits of information that the helper is allowed to give the player is bounded. Even if this is enough information for the player to quickly solve one instance of the problem, the hope was that the helper would not be able to encode the *relevant* information about many instances into the same few bits. What was expected was that if the number of game instances doubles, then the number of bits of help would need to double as well in order for the complexity per instance to remain constant. This, however, is not the case. If the same bits of help are given for an arbitrarily large number of instances, then the complexity of each instance can drop as effectively as if this many bits of help were given independently to each of the instances. It is surprising that enough information about so many game instances can be encoded into so few bits. This appears to contradict Shannon's laws concerning the encoding of information.

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 [IZ89] on recycling random bits.

For the purpose of proving lower bounds on time-space tradeoffs these results are disappointing. Increasing the number of game instances does not help in the way we hoped. However, the

results are still useful. Though, achieving 2 questions per game instance requires very few help bits, a lot of help bits are required to decrease the number of required questions below this number. Furthermore, this 2 (actually a 1.5) is sufficient of provide the lower bound in Chapter 3.

2.1 The Definition of the Game

The task of one game instance 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 [BRT92] for the purpose of proving lower bounds for *st*-connectivity. The basic idea is that the input graph has two identical halves connected by r switchable edges. A vector $\alpha \in \{0,1\}^r$ specifies which of the switchable edges have both ends within the same half of the graph and which span from one half to the other. Traversing from distinguished node s to distinguished node t requires traversing a pebble from one half of the graph to the other. After a pebble traverses a sequence of switchable edges, which half of the graph the pebble is contained in is determined by the parity of the corresponding bits of α. The JAG computation on this graph is modeled by a parity game. The following is a more complex game. A version of it is used in Chapter 3.

The **helper-parity game** with d game instances is defined as follows. There are two parties, a player and a helper. The input 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. These are given to the helper. The helper sends b bits in total about the d vectors. The player 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. For a particular input $\vec{\alpha}$, the complexity of the game is the number of questions asked averaged over the game instances, $c_{\vec{\alpha}} = \frac{1}{d} \sum_{i \in [1..d]} c_{\langle \vec{\alpha}, i \rangle}$, where $c_{\langle \vec{\alpha}, i \rangle}$ is the number of questions asked about the i^{th} game instance on input $\vec{\alpha}$. Both worst case and average case complexities are considered.

One instance of the helper-parity game can be solved using at most $\frac{r}{2^b}$ questions per game instance as follows. Partition the set of input positions $[1..r]$ into 2^b blocks. The helper specifies the first non-zero block. The player asks for the value of each bit within the specified block. A matching worst case complexity lower bound for this problem is easy to obtain.

For d game instances, one might at first believe that the worst case number of questions per game instance would be $\frac{r}{2^{b/d}}$. The justification is that if the helper sends the same b bits to all of the game instances then on average only b/d of these bits would be about the i^{th} game instance. Therefore, the player would need to ask $\frac{r}{2^{b/d}}$ questions about this instance. The helper and the player, however, can do much better than this. The number of questions asked averaged over the number of game instances satisfies

$$\max\left[\ \frac{r}{2^b}; 2 - O\left(\frac{b}{d} + 2^{-r+1}\right)\ \right] \leq \mathrm{Exp}_{\vec{\alpha}} c_{\vec{\alpha}} \leq \max_{\vec{\alpha}} c_{\vec{\alpha}} \leq \max\left[\ \frac{r}{2^b} + \log(e) + \log\left(\frac{2r}{2^b}\right) + o(1);\ 2 + o(1)\ \right].$$

As long as the bound is more than 2, it does not depend on the number of game instances d. Therefore, if we fix the size of each game instance r, fix the number of questions c asked by the player per game instance, and increase the number of game instances d arbitrarily, then the number of help bits b needed does not increase. Given that each player requires some "information" from the helper "about" each game instance, this is surprising. The result $\frac{r}{2^d}$ also says that there is no

difference between the helper independently sending b bits of help for each of the game instances and requiring the helper to send the same b bits for each of the instances.

This chapter is structured as follows. Section 2.2 explains the connection between these results and the work done by Impagliazzo and Zuckerman on recycling random bits [IZ89]. In this light, the results are not as surprising. Section 2.3 proves that a protocol exists that achieves the upper bound and Section 2.4 gives the simple $\frac{r}{2^b}$ lower bound. Section 2.5 presents a recent proof by Rudich [R93] of the $(2-\epsilon)$ lower bound. Section 2.6 proves my own $(2-\epsilon)$ lower bound for the problem under the restriction that the player must solve the game instance being worked on before dynamically choosing the next game instance to work on. Both proofs are presented because they use completely different proof techniques. My proof uses entropy to measure the amount of information the helper's message contains "about" the input α_i to each game instance and then examine how this message partitions the input domain. Rudich avoids the dependency between the game instances caused by the helper's message by using the probabilistic method. Section 2.7 defines a simpler version of the helper-parity game. The st-connectivity lower bound in Chapter 3 could be proved using the original game, but the proof becomes much simpler when using this simpler game. A tight lower bound is proved for this version of the game using my techniques. Rudich's techniques could be used to get the same result.

2.2 Viewing the Helper's Message as Random Bits

The fact that the player need ask only $\max\{\frac{r}{2^b}, 2\}$ questions per game instance may seem surprising. However, if the game is viewed in the right way, it is not surprising at all. The incorrect way of viewing the game is that the helper sends the player information about the input. With this view the result is surprising, because Shannon proves that it requires d times as many bits to send a message about d different independent objects. The correct way of viewing the game is that the helper sends the player purely random bits. With this view the result is not surprising, because Impagliazzo and Zuckerman [IZ89] prove that the player can recycle the random bits used in solving one game instance so that the same random bits can be used to solve other game instances. To understand this, we need to understand how random bits can help the player, understand how a random protocol for the game can be converted into a deterministic protocol, and finally understand how many random bits the helper must send the player.

With no helper bits, if the player asks his questions deterministically, then on the worst case input, he must ask rd questions. However, if the helper sent the player random bits, then the player could deterministically use these to choose random questions (from those that are linearly independent from those previously asked). With random questions, the expected number of questions per a game instance is 2. This is because a random question has odd parity with probability $\frac{1}{2}$.

By the definition of the game, the helper is not able to send a random message, but must send a fixed message $M(\vec{\alpha})$ determined by the input $\vec{\alpha}$. However, this causes no problem for the following reason. Suppose that there is exists a protocol in which the helper sends the players b random bits and on every input $\vec{\alpha}$ there is a non-zero probability that the player succeeds after only c questions. It follows that for every input $\vec{\alpha}$, there exists a fixed message $M(\vec{\alpha})$ for which the player asks so few questions. Hence, a deterministic protocol could be defined that achieves this same complexity by having the helper send this fixed message $M(\vec{\alpha})$ on input $\vec{\alpha}$.

The question remaining is how many random bits a player needs. First consider one game

instance. With b random bits, the player can partition $[1..r]$ into 2^b blocks and ask for the values of each of the bits of α within the randomly specified block. Because the entire vector is non-zero, the specified block will be non-zero with a non-zero probability. In this case, the player will succeed at getting an answer with odd parity. The number of questions asked by the player is $\frac{r}{2^b}$. Hence, $b = \log\left(\frac{r}{c}\right)$ random bits are needed to ensure that the player is able to succeed with non-zero probability after only c questions. Note that this is exactly the complexity of one game instance when viewing the helper's message as containing information about the input. A savings is obtained only when there are many game instances.

Impagliazzo and Zuckerman [IZ89] explain how performing an experiment might require lots of random bits, but the entropy that it "uses" is much less. Their surprising result is that the number of random bits needed for a series of d experiments is only the number of random bits needed for one experiment plus the sum of the entropies that each of the other experiments uses. The consequence of this for the helper-parity game is that if the player recycles the random bits it obtains, then the helper need not send as many bits.

Although this technique decreases the number of helper bits needed, it does not completely explain the amazing upper bound. Recall that the upper bounds says that as the number of game instances increases arbitrarily, the number of helper bits does not need to increase **at all**. The explanation is that we do not require each player to ask fewer than c questions about each game instance. We only require that the total number of questions asked is less than cd. Because at most 2 questions are likely to be asked, this will be the case for most inputs. The player is then able to ask almost $O(cd)$ questions about a few of the game instances and still keep the total number of questions within the requirements. Ensuring that the maximum number of questions asked about a particular instance is at most $O(cd)$ requires far fewer random bits then ensuring that the maximum is c. Therefore, as d increases, the number of random bits needed per game instance decreases so that the total number remains fixed.

Before proving that such a protocol exists, let us try to understand the Impagliazzo and Zuckerman result better by considering a sightly different version of the helper-parity game. Instead of requiring that the total number of questions asked be at most cd, let us consider requiring that for each game instance at most w questions are asked. With this change, the game fits into the Impagliazzo and Zuckerman framework of d separate tasks each with independent measures of success.

This framework needs to know how much entropy is "used up" by each of these game instances. Consider the standard protocol again. The input to the game instance is broken into 2^b pieces of size w. The helper specifies the first block that is non-zero. Because there are 2^b different messages that the helper might send, we assume that the helper must send b bits. However, the entropy of the helper's message (the expected number of bits) is much less than this, because the helper does not send each of its messages with the same probability. For a random input α_i, the first block will be non-zero with probability $1 - 2^{-w}$ and hence the first message will be sent with this probability. The entropy of the message works out to be only $\frac{1}{2^w}$.

The Impagliazzo and Zuckerman's result then says that the number of random bits need for d game instances is $\log\left(\frac{r}{w}\right) + \frac{d}{2^w}$ to give a non-zero probability that all of the game instances succeed after only w questions each. This, in fact, matches the upper and lower bounds for this version of the helper-parity game that I am able to obtain using the same techniques used in Section 2.6.

2.3 The Existence of a Helper-Parity Protocol

This section proves the existence of the protocol that achieves the surprising upper bound.

Theorem 1 *There exists a protocol for the helper-parity game such that for every r and b, as d increases, the number of questions per game instance is at most*

$$\max_{\vec{\alpha}} \left[\frac{1}{d} \sum_{i \in [1..d]} c_{\langle \vec{\alpha}, i \rangle} \right] \leq \max \left[\frac{r}{2^b} + \log(e) + \log\left(\frac{2r}{2^b} \right) + o(1);\ 2 + o(1) \right].$$

Proof of Theorem 1: A protocol will be found for which the number of questions per game instance is at most $c = \max\left[\frac{r}{2^b} + \log\left(e\ \epsilon^{-1/d} \right) + \log\left(\frac{2r}{2^b} + 2\log\left(e\ \epsilon^{-1/d} \right) \right); 2 + \epsilon \right]$, where $\epsilon = 2^{-\sqrt{dr/2^b}}$. Note that $c \geq \frac{r}{2^b} + \sqrt{\frac{r}{d2^b}} + \log(e) + \log\left(\frac{2r}{2^b} + 2\sqrt{\frac{r}{d2^b}} \right)$ and $c \geq 2 + 2^{-\sqrt{dr/2^b}}$. Hence, the required bounds are achieved as d gets sufficiently large.

A helper-parity protocol is randomly chosen as follows. For each message $m \in [1..2^b]$ and for each $i \in [1..d]$, independently at random choose an ordered list of r parity questions, $E_{\langle m,i,1 \rangle}, \ldots, E_{\langle m,i,r \rangle} \subseteq [1..r]$. The player asks questions about each of the game instances in turn. If the help message received was m, then the player asks the questions $E_{\langle m,i,1 \rangle}, \ldots, E_{\langle m,i,r \rangle}$ about α_i one at a time until he receives an answer of 1. For each $\vec{\alpha} \in (\{0,1\}^r - \{0^r\})^d$, let $C_{\langle \vec{\alpha}, m \rangle}$ be the total number of questions that the player asks. On input $\vec{\alpha}$, the helper sends the message $m = M(\vec{\alpha})$, that minimizes this number. In order to prove that there exists a protocol such that $\max_{\vec{\alpha}} \sum_{i \in [1..d]} c_{\langle \vec{\alpha}, i \rangle} \leq cd$, it is sufficient, by the probabilistic method, to prove that $Pr\left[\exists \vec{\alpha}, \forall m, C_{\langle \vec{\alpha}, m \rangle} > cd \right] < 1$.

Fix an input $\vec{\alpha}$ and a message m and consider the random variable $C_{\langle \vec{\alpha}, m \rangle}$ determined by the randomly chosen protocol. Every question asked has a probability of $\frac{1}{2}$ of having an odd parity and the player stops asking questions when he receives d odd parities. Thus $Pr\left[C_{\langle \vec{\alpha}, m \rangle} = k \right]$ is the probability that the d^{th} successful Bernoulli trial occurs on the k^{th} trial. It is distributed according to a negative binomial distribution. Standard probability texts show this probability to be $Pr\left[C_{\langle \vec{\alpha}, m \rangle} = k \right] = \binom{k-1}{d-1} 2^{-k}$.

If the protocol "fails", it must ask some number of questions greater than cd. The probability of this is $Pr\left[C_{\langle \vec{\alpha}, m \rangle} > cd \right] = \sum_{k > cd} \binom{k-1}{d-1} 2^{-k}$. This is the tail of a negative binomial distribution. After $k \geq (2+\epsilon)\, d$, these terms become geometric. Specifically, the ratio of the k^{th} and the $k+1^{st}$ terms is bounded by a constant. Namely,

$$\begin{aligned} \frac{\binom{k}{d-1} 2^{-(k+1)}}{\binom{k-1}{d-1} 2^{-k}} &= \frac{k(k-1)\ldots(k-d+2)}{(k-1)(k-2)\ldots(k-d+1)} 2^{-1} \\ &= \left(\frac{k}{(k-d+1)} \right) \frac{1}{2} \leq \left(\frac{k}{k - \frac{k}{2+\epsilon}} \right) \frac{1}{2} \leq \left(\frac{2+\epsilon}{1+\epsilon} \right) \frac{1}{2} \leq 1 - \frac{\epsilon}{4}. \end{aligned}$$

Note that $A + \left(1 - \frac{\epsilon}{4}\right) A + \left(1 - \frac{\epsilon}{4}\right)^2 A + \ldots = \frac{4}{\epsilon} A$. Therefore, because $cd \geq (2+\epsilon)\, d$, then

$$Pr\left[C_{\langle \vec{\alpha}, m \rangle} > cd \right] \leq \frac{4}{\epsilon} \binom{cd}{d-1} 2^{-(cd+1)} \leq \frac{(cd)^{d-1}}{\epsilon\ (d-1)!} 2^{-cd}$$

$$\leq \frac{(cd)^d}{\epsilon d!} 2^{-cd} \leq \frac{(cd)^d}{\epsilon \left(\frac{d}{e}\right)^d} 2^{-cd} = \left(\frac{ce}{2^c \epsilon^{1/d}}\right)^d .$$

Because there are 2^b messages m, there are 2^b independent chances that the bound on the number of questions succeeds. Therefore, the probability that all 2^b fail is $Pr\left[\forall m, C_{\langle \vec{\alpha}, m \rangle} > cd\right] \leq \left[\left(\frac{ce}{2^c \epsilon^{1/d}}\right)^d\right]^{2^b}$. However, the algorithm must work for each of the $(2^r - 1)^d < 2^{rd}$ inputs $\vec{\alpha}$. Therefore,

$$Pr\left[\exists \vec{\alpha}, \forall m, C_{\langle \vec{\alpha}, m \rangle} > cd\right] < 2^{rd} \left(\frac{ce}{2^c \epsilon^{1/d}}\right)^{d2^b} .$$

Substituting in $c \geq \frac{r}{2^b} + \log\left(e\ \epsilon^{-1/d}\right) + \log\left(\frac{2r}{2^b} + 2\log\left(e\ \epsilon^{-1/d}\right)\right)$ gives

$$2^{rd} \left(\frac{ce}{2^c \epsilon^{1/d}}\right)^{d2^b} \leq 2^{rd} \left(\frac{\left[\frac{r}{2^b} + \log\left(e\ \epsilon^{-1/d}\right) + \log\left(\frac{2r}{2^b} + 2\log\left(e\ \epsilon^{-1/d}\right)\right)\right] e}{2^{\frac{r}{2^b}} \times e\ \epsilon^{-1/d} \times 2\left(\frac{r}{2^b} + \log\left(e\ \epsilon^{-1/d}\right)\right) \times \epsilon^{1/d}}\right)^{d2^b}$$

$$< 2^{rd} \left(\frac{1}{2^{\frac{r}{2^b}}}\right)^{d2^b} = 1.$$

We can conclude there exists a protocol. ■

2.4 The $(r/2^b)$ Lower Bound

The following is a worst case lower bound for the helper-parity game. It is tight for the case when at least 2 questions are asked per game instance.

Theorem 2 *In any protocol for helper-parity game, the average number of questions per game instance is* $\max_{\vec{\alpha}} \left[\frac{1}{d} \sum_{i \in [1..d]} c_{\langle \vec{\alpha}, i \rangle}\right] \geq \frac{r}{2^b}$.

Proof of Theorem 2: First consider $d = 1$ game instance and $b = 0$ bits of help. Suppose that there is a protocol in which at most $r - 1$ questions must be asked. The protocol consists of a list of questions $E_1, \ldots, E_{r-1}$ that will be asked until an answer of 1 is obtained. The set of homogeneous equations $\langle E_1, \ldots, E_{r-1} \rangle^T \alpha = \langle 0, \ldots, 0 \rangle^T$ has a non-zero solution $\alpha \in \{0,1\}^r$. This vector has parity 0 on each of the subsets E_i. Therefore, on input α, the protocol does not get an answer of 1, contrary to the assumptions. Thus, every protocol must ask at least r questions in the worst case.

Now consider d game instances played in parallel with $b = 0$ bits of help. Any input from $(\{0,1\}^r - \{0^r\})^d$ is possible when the player starts asking questions, because the helper provides no information. The player can ask questions about only one of the game instances at a time, so the set of possible inputs always remains in the form of a cross product of d sets. In effect, the lower bound for one game instance can be applied to each of the game instances in parallel. It follows that rd questions are needed in total.

Now consider an arbitrary number of helper bits b. For every subset of vectors $S \subseteq (\{0,1\}^r - \{0^r\})^d$, define $C(S)$ to be the minimum number of questions that the player must ask assuming that the

helper specifies that the input vector $\vec{\alpha}$ is in S. The $b = 0$ case proves that $C\left(\left(\{0,1\}^r - \{0^r\}\right)^d\right) = rd$. The b bits of information sent by the helper partitions the original set of $\left(\{0,1\}^r - \{0^r\}\right)^d$ input vectors into 2^b sets S_m corresponding to each message m. Given a good player questioning protocol for each S_m, simply asking the union of the 2^b sets of questions works for any input because every input is in one of these sets. Thus $\sum_{m \in \{0,1\}^b} C\left(S_m\right) \geq C\left(\left(\{0,1\}^r - \{0^r\}\right)^d\right) = rd$. Therefore, there exists a message m such that $C\left(S_m\right) \geq \frac{rd}{2^b}$. ■

2.5 The $(2-\epsilon)$ Lower Bound Using Probabilistic Techniques

The following is an expected case lower bound for the helper-parity game. It is applies when fewer than 2 questions are asked per game instance. The proof uses the probabilistic method and is by Rudich [R93].

Theorem 3 *For every $\epsilon > 0$, if $b \leq \left(0.34\epsilon^2 - 2^{-r+1}\right) d - \log_2\left(\frac{100}{\epsilon}\right)$, then $Exp_{\vec{\alpha}}\left[\frac{1}{d}\left(\sum_{i \in [1..d]} c_{\langle \vec{\alpha}, i \rangle}\right)\right] \geq 2 - \epsilon$.*

Proof of Theorem 3 [R93]: Consider a helper parity protocol. For each helper's message $m \in \{0,1\}^b$, define $S_m \subseteq \left(\{0,1\}^r - \{0^r\}\right)^d$ to contain those inputs $\vec{\alpha}$ for which the player, after receiving m, asks at most $(2 - 0.98\epsilon)\, d$ questions. We will prove that S_m contains at most a $0.01\epsilon 2^{-b}$ fraction of the inputs $\vec{\alpha} \in \left(\{0,1\}^r - \{0^r\}\right)^d$. There are a most 2^b different help messages $m \in \{0,1\}^b$. Therefore, unioned over all messages, the player asks at most $(2 - 0.98\epsilon)\, d$ questions for at most a 0.01ϵ fraction of the inputs. Even if the player asks zero questions for these inputs and the stated $(2 - 0.98\epsilon)\, d$ questions for the other inputs, the expected number of questions is still at least $(1 - 0.01\epsilon) \times (2 - 0.98\epsilon)\, d \geq (2 - \epsilon)\, d$.

Fix a helper message m. Randomly choose an input $\vec{\alpha}$ uniformly over all the inputs in $\left(\{0,1\}^r\right)^d$. 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. Given any parity question, $\bigoplus_{j \in E} [\alpha_i]_j$ for $E \subseteq [1..r]$ and $i \in [1..d]$, the probability that the answer is odd is exactly $\frac{1}{2}$. After the player receives the answer to a number of parity questions, the probability distribution is restricted to those inputs consistent with the answers obtained. The conditional probability that the next question has an odd answer is still, however, $\frac{1}{2}$. This is, of course, under the reasonable assumption that the questions asked by the player are linearly independent (if not the probability is 0).

After $(2 - 0.98\epsilon)\, d$ questions have been asked, we stop the protocol. Even though the actual questions asked might be chosen dynamically depending on the answers to the previous questions, the probability of getting an odd parity is $\frac{1}{2}$ for each question. Hence, each question can be viewed as an independent trial with $\frac{1}{2}$ probability of success and we are able to apply Chernoff's bound [Sp92]. Let x_i, $i \in [1..n]$ be mutually independent random variables with $\Pr[x_i = 1] = \Pr[x_i = 0] = \frac{1}{2}$, then for every $a > 0$, $\Pr\left[\sum_{i \in [1..n]} x_i \; - \frac{n}{2} > a\right] < e^{\frac{-2a^2}{n}}$.

In our situation, the number of trials is $n = (2 - 0.98\epsilon)\, d$. The player requires at least d odd answers, (one for each game instance). In other words, he requires $\sum_{i \in [1..n]} x_i > d$ which is equivalent to $\sum_{i \in [1..n]} x_i \; - \frac{n}{2} > d - \frac{(2 - 0.98\epsilon)d}{2} = .49\epsilon d$. Chernoff's bound gives that the probability of this occurring is less than $e^{\frac{-2(.49\epsilon d)^2}{(2 - 0.98\epsilon)d}} \leq 2^{-0.34\epsilon^2 d}$. Therefore, for at most this fraction of vectors

$\vec{\alpha} \in (\{0,1\}^r)^d$ does the player obtain d odd answers after only $(2-0.98\epsilon)\,d$ questions.

Since S_m contains at most a $2^{-0.34\epsilon^2 d}$ fraction of the vectors $\vec{\alpha} \in (\{0,1\}^r)^d$, it contains at most a $2^{-0.34\epsilon^2 d} \times \frac{2^{rd}}{(2^r-1)^d}$ fraction of the inputs $\vec{\alpha} \in (\{0,1\}^r - \{0^r\})^d$. Now observe that $1 - x \geq 2^{-2x}$, as long as $x \in [0, \frac{1}{2}]$. Therefore, $\frac{2^{rd}}{(2^r-1)^d} = (1-2^{-r})^{-d} \leq 2^{2^{-r+1}d}$. By the given restriction on the number of help bits b, $2^{(-0.34\epsilon^2+2^{-r+1})d} \leq 0.01\epsilon 2^{-b}$. In conclusion, S_m contains at most a $0.01\epsilon 2^{-b}$ fraction of the inputs $\vec{\alpha} \in (\{0,1\}^r - \{0^r\})^d$. ■

2.6 A $(2-\epsilon)$ Lower Bound Using Entropy for a Restricted Ordering

The following is my original lower bound for the helper-parity game. The proof is more complex than Rudich's proof and applies to a restricted game, but is included because the techniques are completely different and it provides more intuition into the message sent by the helper.

We will say that the d game instances are played in *series* if the player must solve the game instances within continuous blocks of time. He is allowed to dynamically choose the order in which to solve them. However, after starting to ask questions about one instance, he may not start asking about another until an odd answer has been obtained about the first. To help the player, he is given the full input α_i to the game instances that he has finished. The following is a lower bound for this version of the helper-parity game.

Theorem 4 *For every $\epsilon > 0$, there exists a constant z_ϵ such that, in any protocol for d helper-parity game instances played in series,*

$$Exp_{\vec{\alpha}}\left[\frac{1}{d}\left(\sum_{i\in[1..d]} c_{\langle\vec{\alpha},i\rangle}\right)\right] \geq 2 - \epsilon - z_\epsilon\left(\frac{b}{d} + 2^{-r+1}\right).$$

In Section 2.2, it was suggested that the helper need not provided information about the input $\vec{\alpha}$, but need only send random bits. Without help, the player must ask r questions per game instance in the worst case. With random bits, the player can expect to ask only 2 questions per game instance. These random bits can be recycled for the different game instances, hence the helper need send very few bits to obtain this complexity. Clearly, this method does not work if the player wants to ask fewer than 2 questions per game instance. In this case, the helper must reveal to the player a great deal of information about the input. This section uses entropy to measure the amount of information that the helper needs to send "about" each game instance.

Shannon's entropy $H(x)$ of the random variable x measures the expected number of bits of information "contained" in x. In other words, it is the expected number of bits needed to specify which specific value the random variable has. Formally, it is defined to be $H(x) = -\sum_\alpha \Pr[x=\alpha]\log(\Pr[x=\alpha])$. In addition, Shannon's entropy can be used to measure the expected number of bits of information that the random variable x contains "about" the random variable y. This is defined to be $I(x;y) = H(x) + H(y) - H(x,y)$. $H(x,y)$ is the entropy of the joint random value $\langle x, y\rangle$.

The proof of Theorem 4 is structured as follows. Consider a fixed protocol for the d instances of the parity game played in series. Because the game instance are worked on during continuous blocks of time, the protocol can easily be partitioned into disjoint protocols for the d separate game

instances. It is then proved that, on average, only a $\frac{1}{d}$ fraction of the b bits sent by the helper are "about" any fixed game instance. Finally, it is proved that $\frac{b}{d}$ bits of information are not enough for the player to achieve the desired number of questions about this game instance.

The protocol is partitioned into disjoint protocols by considering d players instead of just one. The i^{th} player asks questions only about the i^{th} game instance α_i. On input $\vec{\alpha}$, the helper, who is omnipotent, sends the i^{th} player the information $M_i(\vec{\alpha})$ that he would know just before the question about the i^{th} game instance is asked.

We do not know how much information the player has learned about α_j, by the time he has completed the j^{th} game instance. Therefore, it is easier to assume that he knows the entire vector α_j. This means that the message $M_i(\vec{\alpha})$ must include $M(\vec{\alpha})$ and all the game instance asked about prior to α_i. The situation is complicated by the fact that the order in which the questions are asked is not fixed, but may be chosen dynamically by the player. For example, suppose that for input $\vec{\alpha}$, the helper's message $M(\vec{\alpha})$ causes the protocol to ask about α_1 first. In such a case, $M_1(\vec{\alpha})$ consists of only $M(\vec{\alpha})$. However, if $M(\vec{\alpha}')$ causes the protocol to ask about α'_3 first and $\langle M(\vec{\alpha}'), \alpha'_3 \rangle$ causes the protocol to ask about α'_8 next and $\langle M(\vec{\alpha}'), \alpha'_3, \alpha'_8 \rangle$ causes the protocol to ask about α'_1 next, then $M_1(\vec{\alpha}') = \langle M(\vec{\alpha}'), \alpha'_3, \alpha'_8 \rangle$.

Shannon's entropy provides a measure of the expected number of bits of information the i^{th} player learns "about" his input α_i from this message $M_i(\vec{\alpha})$. Clearly, the more information he has, the better chance he has to ask a question with odd parity. The following lemma states that on average only a $\frac{1}{d}$ fraction of the b bits sent by the helper are "about" any one of the game instances.

Lemma 1 *Let $\vec{x}$ be a random variable chosen uniformly from $(\{0,1\}^r - \{0^r\})^d$.*
$\sum_{i \in [1..d]} I(M_i(\vec{x}); x_i) = I(M(\vec{x}); \vec{x})$.

Corollary 5 $\sum_{i \in [1..d]} I(M_i(\vec{x}); x_i) \leq b$.

Proof of Corollary 5: Note that $I(M; \vec{x}) \leq b$ follows trivially from fact that the helper's message $M(\vec{x})$ contains only b bits. ■

Proof of Lemma 1: By definition, $H(M(\vec{x})) = \frac{-1}{u} \sum_{\vec{\alpha} \in (\{0,1\}^r - \{0^r\})^d} \log(\Pr[M(\vec{x}) = M(\vec{\alpha})])$, where $u = |(\{0,1\}^r - \{0^r\})^d|$. Hence, the right hand side becomes

$$I(M(\vec{x}); \vec{x}) = H(M(\vec{x})) + H(\vec{x}) - H(M(\vec{x}), \vec{x}) = \frac{1}{u} \sum_{\vec{\alpha}} \log\left(\frac{\Pr[M(\vec{x}) = M(\vec{\alpha}) \text{ and } \vec{x} = \vec{\alpha}]}{\Pr[M(\vec{x}) = M(\vec{\alpha})] \Pr[\vec{x} = \vec{\alpha}]} \right)$$

and the left hand side becomes

$$\sum_{i \in [1..d]} I(M_i(\vec{x}); x_i) = \sum_{i \in [1..d]} \frac{1}{u} \sum_{\vec{\alpha}} \log\left(\frac{\Pr[M_i(\vec{x}) = M_i(\vec{\alpha}) \text{ and } x_i = \alpha_i]}{\Pr[M_i(\vec{x}) = M_i(\vec{\alpha})] \Pr[x_i = \alpha_i]} \right).$$

We will equate these separately for each $\vec{\alpha}$. For a fixed input $\vec{\alpha}$ the order in which the questions are asked about the game instances is fixed. Let $\alpha_{\pi_1}, \alpha_{\pi_2}, \ldots, \alpha_{\pi_d}$ be this order. Therefore,

the information that the $\pi_{j+1}{}^{st}$ player receives is $M_{\pi_{j+1}}(\vec{\alpha}) = \left\langle M(\vec{\alpha}), \alpha_{\pi_1}, \ldots, \alpha_{\pi_j} \right\rangle$. In particular, this implies that $\Pr\left[M_{\pi_{j+1}}(\vec{x}) = M_{\pi_{j+1}}(\vec{\alpha})\right] = \Pr\left[M_{\pi_j}(\vec{x}) = M_{\pi_j}(\vec{\alpha}) \quad \text{and} \quad x_{\pi_j} = \alpha_{\pi_j}\right]$, where $M_{\pi_{d+1}}(\vec{\alpha}) = \langle M(\vec{\alpha}), \alpha_{\pi_1}, \ldots, \alpha_{\pi_d} \rangle$. Hence,

$$
\begin{aligned}
&\sum_{i \in [1..d]} \log\left(\frac{\Pr\left[M_i(\vec{x}) = M_i(\vec{\alpha}) \quad \text{and} \quad x_i = \alpha_i\right]}{\Pr\left[M_i(\vec{x}) = M_i(\vec{\alpha})\right] \Pr\left[x_i = \alpha_i\right]}\right) \\
&= \sum_{j \in [1..d]} \log\left(\frac{\Pr\left[M_{\pi_j}(\vec{x}) = M_{\pi_j}(\vec{\alpha}) \quad \text{and} \quad x_{\pi_j} = \alpha_{\pi_j}\right]}{\Pr\left[M_{\pi_j}(\vec{x}) = M_{\pi_j}(\vec{\alpha})\right] \Pr\left[x_{\pi_j} = \alpha_{\pi_j}\right]}\right) \\
&= \sum_{j \in [1..d]} \log\left(\frac{\Pr\left[M_{\pi_{j+1}}(\vec{x}) = M_{\pi_{j+1}}(\vec{\alpha})\right]}{\Pr\left[M_{\pi_j}(\vec{x}) = M_{\pi_j}(\vec{\alpha})\right] \Pr\left[x_{\pi_j} = \alpha_{\pi_j}\right]}\right) \\
&= \log\left(\frac{\Pr\left[M_{\pi_{d+1}}(\vec{x}) = M_{\pi_{d+1}}(\vec{\alpha})\right]}{\Pr\left[M_{\pi_1}(\vec{x}) = M_{\pi_1}(\vec{\alpha})\right] \Pi_{i \in [1..d]} \Pr\left[x_{\pi_i} = \alpha_{\pi_i}\right]}\right) \\
&= \frac{1}{u} \log\left(\frac{\Pr\left[M(\vec{x}) = M(\vec{\alpha}) \quad \text{and} \quad \vec{x} = \vec{\alpha}\right]}{\Pr\left[M(\vec{x}) = M(\vec{\alpha})\right] \Pr\left[\vec{x} = \vec{\alpha}\right]}\right)
\end{aligned}
$$

The lemma follows. ■

The protocol has now been partitioned into disjoint protocols for the d separate game instances and it has been proven that on average $\frac{b}{d}$ bits of help has been provided to each separate protocol. What remains to prove is that this is not enough information for a particular player to effectively ask questions about his game instance.

Lemma 2 *For every $\epsilon > 0$, there exists a constant z_ϵ such that, in any protocol for one helper-parity game instance*
$Exp_{\vec{\alpha}}\left[c_{\langle \vec{\alpha}, i \rangle}\right] \geq 2 - \epsilon - z_\epsilon \left(I\left(M_i(\vec{x}); x_i\right) + 2^{-r+1}\right)$.

Before proving the lemma, the following minimization property is needed.

Claim 1 *Let z be a constant and let $\{q_i\}$ and $\{e_i\}$ be finite sets of constants. Define the set of constants $\{q'_i\}$ such that $\sum_i q'_i = \sum_i q_i$ and $q'_i = \left(\frac{\mu}{e_i}\right)^{\frac{1}{z}}$ for some constant μ. It follows that $\sum_i q_i \log\left(e_i q_i^z\right) \geq \sum_i q'_i \log\left(e_i (q'_i)^z\right) = \sum_i q'_i \log(\mu)$.*

Proof of Claim 1: By way of contradiction, suppose that the values $\{q_i\}$ minimize the sum $\sum_i q_i \log\left(e_i q_i^z\right)$ and that there exists indexes j and k such that $e_j q_j^z \neq e_k q_k^z$. We will now minimize the summation subject to changing q_j and q_k, but leaving all the other q_i's fixed. Because the condition $\sum_i q_i = K$ must be maintained, there exists a constant K' for which $q_k = K' - q_j$. The equation to minimize becomes $y = K'' + q_j \log\left(e_j q_j^z\right) + (K' - q_j) \log\left(e_k (K' - q_j)^z\right)$. Setting the derivative to zero gives $\frac{dy}{dq_j} = \log\left(e_j q_j^z\right) + \frac{q_j}{e_j q_j^z} z e_j q_j^{z-1} - \log\left(e_k (K' - q_j)^z\right) + \frac{K' - q_j}{e_k (K' - q_j)^z} z e_k (K' - q_j)^{z-1} (-1) = 0$. Solving gives $\log\left(e_j q_j^z\right) = \log\left(e_k (K' - q_j)^z\right) = \log\left(e_k q_k^z\right)$. Therefore, the minimum occurs when $e_j q_j^z = e_k q_k^z$, contradicting the above assumptions. ■

Proof of Lemma 2: Consider a protocol. For each message $M_i(\vec{\alpha}) = m$ sent by the helper to the i^{th} player, the player asks a series of parity questions about the i^{th} game instance α_i and stops when he receives an answer of 1. Let $E^{\langle m,1\rangle}, E^{\langle m,2\rangle}, \ldots \subseteq [1..r]$ denote the parity questions asked.

In order to simplify the notation, let $\beta = \langle \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_d \rangle$ and hence $\vec{\alpha} = \langle \alpha_i, \beta \rangle$. The message sent by the helper partitions the universe of possible inputs $\langle \alpha_i, \beta \rangle$. Let $u = |(\{0,1\}^r - \{0^r\})^d|$, $s_m = |\{\langle \alpha_i, \beta \rangle \mid m = M_i(\alpha_i, \beta)\}|$, $A_m = \{\alpha_i \mid \exists \beta, m = M_i(\alpha_i, \beta)\}$, $Q_{\langle m,\alpha_i \rangle} = \{\beta \mid m = M_i(\alpha_i, \beta)\}$, and $q_{\langle m,\alpha_i \rangle} = |Q_{\langle m,\alpha_i \rangle}|$. By definition, $\sum_{\alpha_i} q_{\langle m,\alpha_i \rangle} = s_m$.

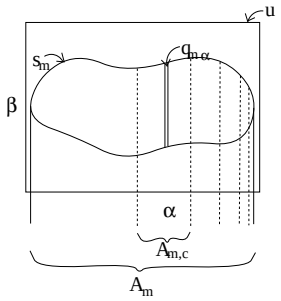

Figure 2.1: The helper's message partitions the input space

Consider a message $M_i(\vec{\alpha}) = m$ sent by the helper to the i^{th} player. After receiving this message the i^{th} player asks the parity questions $E^{\langle m,1\rangle}, E^{\langle m,2\rangle}, \ldots \subseteq [1..r]$ in order until an answer of 1 is obtained. Define $c_{\langle m,\alpha_i \rangle}$ to be the number of the $E^{\langle m,1\rangle}, E^{\langle m,2\rangle}, \ldots$ questions asked when the player has input α_i. Recall that A_m is the set of inputs α_i for which the player might receive the message m. Sort the inputs in A_m in increasing order according to $c_{\langle m,\alpha_i \rangle}$ and partition the list into w_m parts $A_{\langle m,1\rangle}, \ldots, A_{\langle m,w_m\rangle}$ of geometrically decreasing size so that $\forall c \in [1..w_m], |A_{\langle m,c\rangle}| = \frac{2^r}{2^c}$. (Except maybe the last, which is exponentially small.) I claim that $\forall c \in [1..w_m], \forall \alpha_i \in A_{\langle m,c\rangle}, c_{\langle m,\alpha_i \rangle} \geq c$. This is because the number of r-bit vectors α_i' for which one of the questions $E^{m,1}, \ldots, E^{m,c-1}$ has parity 1 is $\sum_{j \in [1..c-1]} \frac{2^r}{2^j}$ and there are at least this many vectors proceeding α_i in the sorted list. Given this notation, the expected number of questions asked can be expressed as

$$\begin{aligned}
\mathrm{Exp}_{\alpha_i,\beta}\left(c_{\langle M_i(\alpha_i,\beta),\alpha_i\rangle}\right) &= \frac{1}{u}\sum_m \sum_{c\in[1..w_m]} \sum_{\alpha_i \in A_{\langle m,c\rangle}} \sum_{\beta \in Q_{\langle m,\alpha_i\rangle}} c_{\langle m,\alpha_i\rangle} \\
&\geq \frac{1}{u}\sum_m \sum_{c\in[1..w_m]} \sum_{\alpha_i \in A_{\langle m,c\rangle}} q_{\langle m,\alpha_i\rangle} c.
\end{aligned}$$

As well, we can express

$$\begin{aligned}
I(M_i(\vec{x}); x_i) &= \frac{1}{u}\sum_{\alpha_i,\beta} \log\left(\frac{\Pr[M_i(\vec{x}) = M_i(\vec{\alpha}) \text{ and } x_i = \alpha_i]}{\Pr[M_i(\vec{x}) = M_i(\vec{\alpha})]\Pr[x_i = \alpha_i]}\right) \\
&= \frac{1}{u}\sum_{\alpha_i,\beta} \log\left(\frac{q_{\langle m,\alpha_i\rangle}/u}{(s_m/u)(1/|\{\alpha_i\}|)}\right) \\
&= \frac{1}{u}\sum_m \sum_{c\in[1..w_m]} \sum_{\alpha_i \in A_{\langle m,c\rangle}} q_{\langle m,\alpha_i\rangle} \log\left(\frac{(2^r-1)\, q_{\langle m,\alpha_i\rangle}}{s_m}\right).
\end{aligned}$$

Let z be a constant to be chosen later. Combining the above two expressions gives

$$\mathrm{Exp}_{\alpha_i,\beta}\left(c_{\langle M_i(\alpha_i,\beta),\alpha_i\rangle}\right) + zI(M_i(\vec{x}); x_i)$$

$$= \frac{1}{u}\sum_m \sum_{c\in[1..w_m]} \sum_{\alpha_i \in A_{\langle m,c\rangle}} q_{\langle m,\alpha_i\rangle} \log\left(2^c \left(\frac{(2^r-1)\, q_{\langle m,\alpha_i\rangle}}{s_m}\right)^z\right).$$

In order to bound this amount, consider each m separately. For all c and all $\alpha_i \in A_{\langle m,c\rangle}$, define $q'_{\langle m,\alpha_i\rangle}$ such that $\sum_{c\in[1..w_m]} \sum_{\alpha_i\in A_{\langle m,c\rangle}} q'_{\langle m,\alpha_i\rangle} = \sum_{c\in[1..w_m]} \sum_{\alpha_i\in A_{\langle m,c\rangle}} q_{\langle m,\alpha_i\rangle}$ and $q'_{\langle m,\alpha_i\rangle} = \left(\frac{\mu_m}{2^c}\right)^{\frac{1}{z}} \frac{s_m}{(2^r-1)}$, for some constant μ_m. (i.e. the argument of the logarithm $\mu_m = 2^c \left(\frac{(2^r-1)q'_{\langle m,\alpha_i\rangle}}{s_m}\right)^z$ is a constant). By Claim 1, it is clear that

$$\begin{aligned}
&\mathrm{Exp}_{\alpha_i,\beta}\left(c_{\langle M_i(\alpha_i,\beta),\alpha_i\rangle}\right) + zI\left(M_i\left(\vec{x}\right); x_i\right) \\
&\leq \frac{1}{u}\sum_m \sum_{c\in[1..w_m]} \sum_{\alpha_i\in A_{\langle m,c\rangle}} q'_{\langle m,\alpha_i\rangle} \log\left(2^c \left(\frac{(2^r-1)\, q'_{\langle m,\alpha_i\rangle}}{s_m}\right)^z\right).
\end{aligned}$$

We need to determine what this constant μ_m is given the constraints on it. The first constraint insures that that $s_m = \sum_{c\in[1..w_m]} \sum_{\alpha_i\in A_{\langle m,c\rangle}} q'_{\langle m,\alpha_i\rangle}$. Substituting in the second constraint gives that

$$s_m = \sum_{c\in[1..w_m]} \sum_{\alpha_i\in A_{\langle m,c\rangle}} \left(\frac{\mu_m}{2^c}\right)^{\frac{1}{z}} \frac{s_m}{(2^r-1)}.$$

The terms no longer depend on α_i. In addition, $|A_{\langle m,c\rangle}| = \frac{2^r}{2^c}$ for each $c \in [1..w_m]$. Therefore,

$$\begin{aligned}
s_m &= \sum_{c\in[1..w_m]} \frac{2^r}{2^c}\left(\frac{\mu_m}{2^c}\right)^{\frac{1}{z}} \frac{s_m}{(2^r-1)} = \frac{2^r}{2^r-1}\left(\mu_m\right)^{\frac{1}{z}} s_m \sum_{c\in[1..w_m]} \left[2^{1+\frac{1}{z}}\right]^{-c} \\
&= \frac{2^r}{2^r-1}\left(\mu_m\right)^{\frac{1}{z}} s_m \left[\frac{1-2^{-w_m\left(1+\frac{1}{z}\right)}}{\left(2^{1+\frac{1}{z}}\right)-1}\right].
\end{aligned}$$

Solving for μ_m gives

$$\begin{aligned}
\mu_m &= \left[\frac{2^r-1}{2^r}\frac{\left(2^{1+\frac{1}{z}}\right)-1}{1-2^{-w_m\left(1+\frac{1}{z}\right)}}\right]^z \text{ and} \\
\log\left(\mu_m\right) &= z\left[\log\left(2^{1+\frac{1}{z}}-1\right) + \log\left(1-2^{-r}\right) - \log\left(1-2^{-w_m\left(1+\frac{1}{z}\right)}\right)\right].
\end{aligned}$$

Now observe that $1-x \geq 2^{-2x}$ as long as $x \in [0, \frac{1}{2}]$, since it is true at endpoints and 2^{-2x} is concave. Therefore, $\log\left(1-2^{-r}\right) \geq -2^{-r+1}$. Similarly, the last term $-\log\left(1-2^{-w_m\left(1+\frac{1}{z}\right)}\right)$ is positive and insignificant. This gives $\log\left(\mu_m\right) \geq z\log\left(2^{1+\frac{1}{z}}-1\right) - z2^{-r+1}$. The limit of $z\log\left(2^{1+\frac{1}{z}}-1\right)$ as z goes to infinity is 2. Therefore, for any ϵ there is a constant z_ϵ such that $\log\left(\mu_m\right) \geq 2-\epsilon-z_\epsilon 2^{-r+1}$.

We can now conclude by stating that

$$\begin{aligned}
&\mathrm{Exp}_{\alpha_i,\beta}\left(c_{\langle M_i(\alpha_i,\beta),\alpha_i\rangle}\right) + z_\epsilon I\left(M_i;\alpha_i\right) \\
&\geq \frac{1}{u}\sum_m \sum_{c\in[1..w_m]} \sum_{\alpha_i\in A_{\langle m,c\rangle}} q'_{\langle m,\alpha_i\rangle} \log\left(\mu_m\right) \\
&\geq \left[\frac{1}{u}\sum_m \sum_{c\in[1..w_m]} \sum_{\alpha_i\in A_{\langle m,c\rangle}} \sum_{\beta\in Q_{\langle m,\alpha_i\rangle}} 1\right]\left[\, 2-\epsilon-z_\epsilon 2^{-r+1}\,\right] \\
&= [1]\left[2-\epsilon-z_\epsilon 2^{-r+1}\right]. \quad \blacksquare
\end{aligned}$$

Now that we know that the player does not have enough information to effectively ask questions about an average game instance, we are ready to prove that the total number of questions charged to the player is at least $(2-\epsilon)\,d$.

Proof of Theorem 4:

$$\begin{aligned}
&\mathrm{Exp}_{\vec{\alpha}}\left[\frac{1}{d}\left(\sum_{i\in[1..d]} c_{\langle\vec{\alpha},i\rangle}\right)\right] \\
&= \frac{1}{d}\sum_{i\in[1..d]} \mathrm{Exp}_{\vec{\alpha}}\left[c_{\langle\vec{\alpha},i\rangle}\right] \text{ (by Lemma 2)} \\
&\geq \frac{1}{d}\sum_{i\in[1..d]} \left[2-\epsilon-z_\epsilon\left(I\left(M_i\left(\vec{x}\right);x_i\right)+2^{-r+1}\right)\right] \\
&= 2-\epsilon-z_\epsilon\left(\frac{1}{d}\sum_{i\in[1..d]} I\left(M_i\left(\vec{x}\right);x_i\right)\right)-z_\epsilon\left(2^{-r+1}\right) \text{ (by Lemma 1)} \\
&\geq 2-\epsilon-z_\epsilon\left(\frac{b}{d}+2^{-r+1}\right). \ \blacksquare
\end{aligned}$$

2.7 A Simpler Version of the Helper-Parity Game

This section defines the version of the helper-parity game that is actually used in Chapter 3. It is simpler and hence makes the *st*-connectivity lower bound simpler.

As before, the input consists of a vector for each of the game instances $\alpha_1,\ldots,\alpha_d \in \{0,1\}^r - \{0^r\}$, the helper sends a help message $M\left(\vec{\alpha}\right)$ containing at most b bits of information, and the player asks parity questions until he asks a question with parity 1 for each of the d game instances. However, in this version, the player is charged for at most the first two questions asked about each game instance.

An equivalent formulation of the game is as follows. First, the helper sends the message $M\left(\vec{\alpha}\right)$ to the player. Then, repeatedly, the player selects a game instance i (in any order) and a parity question E_i. If the answer to the question is 1 then he is charged only 1 for the question. If the answer is 0 then he is charged 2. 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 of the game on input $\vec{\alpha}$ is defined to be the total charge per game instance,

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

These two formulations of the game are equivalent because in both cases the cost for a game instance depends only on the answer to the first question. As well, in the second formulation, the player is given the input α_i for free, but in the first formulation the player is not charged if he keeps asking questions until he learns this information himself.

The first formulation makes it clear that the complexity of the original game is at least the complexity of this game. The second formulation simplifies the game, making the reduction from

st-connectivity much easier. Specifically, after the JAG "asks" a first parity question about a subgraph, the structure of this subgraph can be revealed to the JAG. Hence, the proof does not need to be concerned about the model knowing partial information about the subgraph. The proof does go through if this is not done, but it is more complex. Besides, the complexities of the versions are $(2-\epsilon)$ and $(1.5-\epsilon)$ and hence effect the final result by very little.

The worst case complexity for this version of the game without any helper bits is clearly 2. With random bits, the expected complexity is clearly 1.5. This section uses the same techniques used in Section 2.6 to prove that a helper must provide lots of bits of information for the complexity to be significantly less than 1.5.

Theorem 6 *For every $\epsilon > 0$, there exists a $\delta > 0$ and a large enough constant r such that if $b < \delta d$, then $Exp_{\vec{\alpha}} c_{\vec{\alpha}} \geq 1.5 - \epsilon$.*

For example, if $\epsilon = 0.0125, \delta = 0.0002, r = 13$, and $d \geq \frac{1}{\delta} b$, then $c \geq 1.5 - \epsilon$. The proof is presented in Section 2.7.

The structure of the proof is the same as that for Theorem 4. Now, however, a fixed protocol can be partitioned easily into d disjoint protocols, without requiring a restriction on the order the player asks his questions. This is because, in the second formulation, the player asks only one question per game instance.

Lemma 3 *Let P be the probability that the first question the player asks about the i^{th} game instance has even parity. For all $\epsilon > 0$, there exists $\delta > 0$ and r such that if $I(M_i(\vec{x}); x_i) \leq \delta$, then $P \geq \frac{1}{2} - \epsilon$.*

Proof of Lemma 3: As before, let $\beta = \langle \alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_d \rangle$ and $\vec{\alpha} = \langle \alpha_i, \beta \rangle$. For each helper message m, let $E^m \subseteq [1..r]$ be the parity question asked by the player. Partition the domain of $\alpha_i \in \{0,1\}^r - \{0^r\}$ into $F_m = \{\alpha_i \mid \bigoplus_{j \in E^m} [\alpha_i]_j = 0\}$ and $\overline{F}_m = \{\alpha_i \mid \bigoplus_{j \in E^m} [\alpha_i]_j = 1\}$. Clearly, $|F_m| = \frac{2^r}{2} - 1$ and $|\overline{F}_m| = \frac{2^r}{2}$. The message sent by the helper partitions the universe of possible inputs $\langle \alpha_i, \beta \rangle$. Let $u = |(\{0,1\}^r - \{0^r\})^d|$, $s_m = |\{\langle \alpha_i, \beta \rangle \mid m = M_i(\alpha_i, \beta)\}|$, $Q_{\langle m, \alpha_i \rangle} = \{\beta \mid m = M_i(\alpha_i, \beta)\}$, and $q_{\langle m, \alpha_i \rangle} = |Q_{\langle m, \alpha_i \rangle}|$. From these values, we can express and compare P and $I(M_i(\vec{x}); x_i)$.

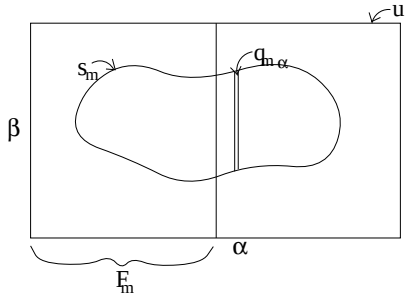

Figure 2.2: The helper's message partitions the input space

Note that the probability that the first question the player asks about the i^{th} game instance

has even parity is

$$P = \frac{1}{u} \sum_{m} \sum_{\alpha_i \in F_m} q_{\langle m, \alpha_i \rangle},$$

because the summation counts the number of $\vec{\alpha} = \langle \alpha_i, \beta \rangle$ for which the parity is 0. Similarly,

$$\begin{aligned}
& I\left(M_i\left(\vec{x}\right); x_i\right) \\
&= \frac{1}{u} \sum_{\alpha_i, \beta} \log \left(\frac{\Pr\left[M_i\left(\vec{x}\right) = M_i\left(\vec{\alpha}\right) \quad \text{and} \quad x_i = \alpha_i\right]}{\Pr\left[M_i\left(\vec{x}\right) = M_i\left(\vec{\alpha}\right)\right] \Pr\left[x_i = \alpha_i\right]} \right) \\
&= \frac{1}{u} \sum_{m} \sum_{\alpha_i} \sum_{\beta \in Q_{\langle m, \alpha_i \rangle}} \log \left(\frac{q_{\langle m, \alpha_i \rangle} / u}{\left(s_m / u\right)\left(1 / \left|\{\alpha_i\}\right|\right)} \right) \\
&= \frac{1}{u} \sum_{m} \sum_{\alpha_i \in F_m} q_{\langle m, \alpha_i \rangle} \log \left(\frac{\left(2^r - 1\right) q_{\langle m, \alpha_i \rangle}}{s_m} \right) + \frac{1}{u} \sum_{m} \sum_{\alpha_i \in \overline{F}_m} q_{\langle m, \alpha_i \rangle} \log \left(\frac{\left(2^r - 1\right) q_{\langle m, \alpha_i \rangle}}{s_m} \right).
\end{aligned}$$

We will bound the two summations separately.

For all m and all $\alpha_i \in F_m$, define $q'_{\langle m, \alpha_i \rangle}$ such that $\sum_m \sum_{\alpha_i \in F_m} q'_{\langle m, \alpha_i \rangle} = \sum_m \sum_{\alpha_i \in F_m} q_{\langle m, \alpha_i \rangle}$ and $q'_{\langle m, \alpha_i \rangle} = \frac{s_m}{\left(2^r - 1\right)} \mu_F$, for some constant μ_F. Similarly, for all m and all $\alpha_i \in \overline{F}_m$, define $q''_{\langle m, \alpha_i \rangle}$ such that $\sum_m \sum_{\alpha_i \in \overline{F}_m} q''_{\langle m, \alpha_i \rangle} = \sum_m \sum_{\alpha_i \in \overline{F}_m} q_{\langle m, \alpha_i \rangle}$ and $q''_{\langle m, \alpha_i \rangle} = \frac{s_m}{\left(2^r - 1\right)} \mu_{\overline{F}}$, for some constant $\mu_{\overline{F}}$. By the first requirement, it is clear that $P = \frac{1}{u} \sum_m \sum_{\alpha_i \in F_m} q'_{\langle m, \alpha_i \rangle}$ and that $1 - P = \frac{1}{u} \sum_m \sum_{\alpha_i \in \overline{F}_m} q''_{\langle m, \alpha_i \rangle}$. As well, by Claim 1, it is clear that

$$\begin{aligned}
& I\left(M_i\left(\vec{x}\right); x_i\right) \\
&\geq \frac{1}{u} \sum_{m} \sum_{\alpha_i \in F_m} q'_{\langle m, \alpha_i \rangle} \log \left(\frac{\left(2^r - 1\right) q'_{\langle m, \alpha_i \rangle}}{s_m} \right) + \frac{1}{u} \sum_{m} \sum_{\alpha_i \in \overline{F}_m} q''_{\langle m, \alpha_i \rangle} \log \left(\frac{\left(2^r - 1\right) q''_{\langle m, \alpha_i \rangle}}{s_m} \right) \\
&= [P] \log\left(\mu_F\right) + [1 - P] \log\left(\mu_{\overline{F}}\right).
\end{aligned}$$

The next step is to bound μ_F and $\mu_{\overline{F}}$ using the fact that $P = \frac{1}{u} \sum_m \sum_{\alpha_i \in F_m} q'_{\langle m, \alpha_i \rangle} = \frac{1}{u} \sum_m \sum_{\alpha_i \in F_m} \frac{s_m}{\left(2^r - 1\right)} \mu_F$. Using $|F_m| = \frac{2^r}{2} - 1$ and $\sum_m s_m = u$, gives $P = \frac{1}{u} \sum_m \left(\frac{2^r}{2} - 1\right) \frac{s_m}{\left(2^r - 1\right)} \mu_F = \frac{\left(\frac{2^r}{2} - 1\right)}{\left(2^r - 1\right)} \mu_F$ or that $\mu_F \geq 2P$. The same calculations for the second summation over $\overline{F}_m$ gives that $\mu_{\overline{F}} = \frac{2^r - 1}{\frac{2^r}{2}} (1 - P) = 2\left(1 - 2^{-r}\right)(1 - P)$. To be consistent, $\mu_F \geq 2\left(1 - 2^{-r}\right) P$.

Returning to I, this gives $I\left(M_i\left(\vec{x}\right); x_i\right) \geq [P] \log\left(2\left(1 - 2^{-r}\right) P\right) + [1 - P] \log\left(2\left(1 - 2^{-r}\right)(1 - P)\right)$ $= 1 + \log\left(1 - 2^{-r}\right) + P \log\left(P\right) + (1 - P) \log\left(1 - P\right)$. Note that $\log\left(1 - 2^{-r}\right) \geq -2^{-r+1}$ as long as $r \geq 1$ and let $U\left(P\right) = 1 + P \log\left(P\right) + (1 - P) \log\left(1 - P\right)$. This gives us that $\left(I\left(M_i; \alpha_i\right) + 2^{-r+1}\right) \geq U\left(P\right)$. Plotting $U\left(P\right)$ reveals the shape of a "U" with key points being $(P, U) = (0, 1), (.5, 0)$, and $(1, 1)$. This means that if $\left(I\left(M_i; \alpha_i\right) + 2^{-r+1}\right)$ is small, then $U\left(P\right)$ is small, and then P is close to $\frac{1}{2}$. We can conclude that $\forall \epsilon > 0$, $\exists \delta > 0$, and a large enough r, such that if $I\left(M_i; \alpha_i\right) \leq \delta$, then $P \geq \frac{1}{2} - \epsilon$. ■

Proof of Theorem 6: Let $\epsilon > 0$ be an arbitrary constant and let $\epsilon' = \frac{\epsilon}{2.5}$. Let δ' and r be the constants proved to exist in Lemma 3 for ϵ'. Let $\delta = \epsilon' \times \delta'$. By Lemma 1, $\sum_{i \in [1..d]} I\left(M_i\left(\vec{x}\right); x_i\right) \leq b \leq \delta d = \epsilon' \delta' d$. It follows that for at least $\left(1 - \epsilon'\right) d$ of the game instances, it is the case that

$I(M_i(\vec{x}); x_i) \leq \delta'$. Hence, for these game instances $Pr\left[\bigoplus_{j \in E_i} [\alpha_i]_j = 0\right] \geq \frac{1}{2} - \epsilon'$. We can conclude that

$$\begin{aligned}
\mathrm{Exp}_{\vec{\alpha}} c_{\vec{\alpha}} &= \mathrm{Exp}_{\vec{\alpha}} \left[\frac{1}{d} \sum_{i \in [1..d]} \left\{ \begin{array}{ll} 1 & \text{if } \bigoplus_{j \in E_i} \left[\alpha_{\langle l,i \rangle}\right]_j = 1 \\ 2 & \text{otherwise} \end{array} \right\} \right] \\
&= \frac{1}{d} \sum_{i \in [1..d]} [1(1 - P_i) + 2P_i] \;=\; \frac{1}{d} \sum_{i \in [1..d]} [1 + P_i] \\
&\geq \frac{1}{d} \times (1 - \epsilon') d \times [1.5 - \epsilon'] \\
&\geq 1.5 - 2.5\epsilon' \;=\; 1.5 - \epsilon. \;\blacksquare
\end{aligned}$$

Chapter 3

Undirected st-Connectivity on a Helper-JAG

In this chapter, a lower bound is proved on the time-space tradeoff for undirected st-connectivity.

Theorem 7 *For every constant $z \geq 2$, the expected time to solve undirected st-connectivity for 3-regular graphs on a probabilistic JAG with $p \leq \frac{1}{28z}\frac{\log n}{\log\log n}$ pebbles and $q \leq 2^{\log^z n}$ states is at least $n \times 2^{\frac{1}{25z}\frac{\log n}{\log\log n}}$.*

See Section 1.1 for a formal definition of a probabilistic JAG. The graphs considered are called "recursive fly swatter graphs" and are built recursively from the squirrel cage graph considered in [BBRRT90]. Their lower bound applies when there is at most one moving pebble. My bound applies as long as there are fewer pebbles than the number of levels of recursion.

The input graph is composed of many fly swatter subgraphs. When a superlinear amount of time is available, the JAG may traverse such a subgraph many times. Although traversing a particular subgraph the first time may take many time steps, the JAG 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 is allowed to set the JAG's initial state and the initial position of the pebbles to minimize the computation time. In this way, the helper is able to communicate to the JAG at least as much information about the input as it could learn during 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 during subsequent traversals.

The helper JAG lower bound is obtained by reducing the st-connectivity problem to the problem of traversing from s to t and then reducing this problem to the **helper-parity game**. The game characterizes the relationship between the number of bits of help (or foreknowledge) and the resulting traversal time. Upper and lower bounds for this and other similar games are found in Chapter 2.

Section 3.1 reduces the st-traversal problem to the st-connectivity problem. Once the computation problem being considered is not a decision problem, I am able to define the helper JAG. This is done in Section 3.2. Section 3.3 describes the fly swatter graph and provides intuition about the complexity of the traversal of this family of graphs. This motivates the helper parity

game defined in Section 2.7. Section 3.4 describes a graph consisting of a line of fly swatters and informally compares the complexity of traversing such a graph to that of the helper parity game. Section 3.5 describes the recursive construction of the input graphs and informally estimates the complexity of its st-traversal. Section 3.6 defines a measure of the time and pebble resources that are used to traverse a particular level of the recursion. Sections 3.3, 3.4, 3.5, and 3.6 do **not** provide a formal proof, but they do give some crucial intuition. Section 3.7 reduces the traversal problem to the helper parity game and Section 3.8 proves the average case lower bound for a natural input distribution. By Yao's theorem, this is sufficient to prove the expected case probabilistic lower bound in the stated theorem.

3.1 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 way 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: 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 4 *If a JAG solves st-connectivity for every graph in $\mathcal{F}'$ (in T steps), then it performs st-traversal for every graph in $\mathcal{F}$ (in T steps).*

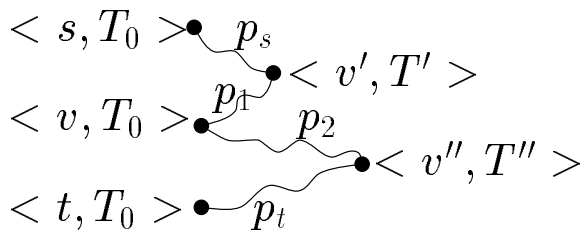

Figure 3.1: A path in the traversal graph of G

Proof of Lemma 4: Suppose that there is a JAG that solves st-connectivity for all graphs in $\mathcal{F}'$. We prove that the JAG must traverse either from s to t or from s' to t'. By way of contradiction, suppose the JAG is given graph $G_{s,t} \cup G_{s',t'}$ as input and it neither traverses 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 the 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. ■

Confusion is added to the lower bound proof by the fact that the path through the traversal graph H may go forwards and backwards in time many times. As demonstrated in the above example, this is caused by the graph initially containing pebbles within the graph and by the pebbles entering the graph via both the distinguished nodes s and t. By the definition of the st-traversal problem, all the pebbles are initially placed on s and t, and not within the graph. However, in the proof of the lower bound, we need to consider the traversal of sub-graphs during sub-intervals of time. By *initially*, I mean at the beginning of the sub-interval of time during which the traversal takes place. Therefore, the pebbles could initially have any placement through the graph, and hence the path through H could be complex.

The following defines the type of traversals for which proving the 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 of 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 inter them via both the distinguished nodes s_i and t_i.

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

Proof of Claim 2: $|S_0| \leq p$, 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.

3.2 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.

In order to prove Theorem 7, it is sufficient to prove the theorem with the following changes. The input domain is restricted to the family of recursive fly swatter graphs. The *st*-traversal problem is considered instead of *st*-connectivity. By Lemma 4, this is sufficient. As well, the helper JAG model is considered instead of the regular JAG. Finally, 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. Yao [Ya77] proves that this is sufficient, since there exists a random algorithm whose expected running time on the worst case input is T iff for **every** distribution on inputs, there exists a deterministic algorithm whose average running time according to the distribution is T. The average case lower bound is, however, a stronger statement because it is proved for a fixed distribution that is natural for the family of graphs being considered.

Theorem 7′ *For every constant $z \geq 2$, the average time for st-traversal by helper JAG with $p \leq \frac{1}{28z}\frac{\log n}{\log\log n}$ pebbles and $q \leq 2^{\log^z n}$ states is at least $n \times 2^{\frac{1}{25z}\frac{\log n}{\log\log n}}$, where the input graph $G(\vec{\alpha}_1, ..., \vec{\alpha}_L)$ is chosen by uniformly choosing $\vec{\alpha}_1, ..., \vec{\alpha}_L$ from $(\{0,1\}^r - \{0^r\})^d$ (for sufficiently large n).*

The family of graphs $G(\vec{\alpha}_1, ..., \vec{\alpha}_L)$ is defined in the next section.

3.3 A Fly Swatter Graph

The basic components of the input graphs defined in Section 3.5 are the **fly swatter graph**. 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 [BBRRT90]. 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.

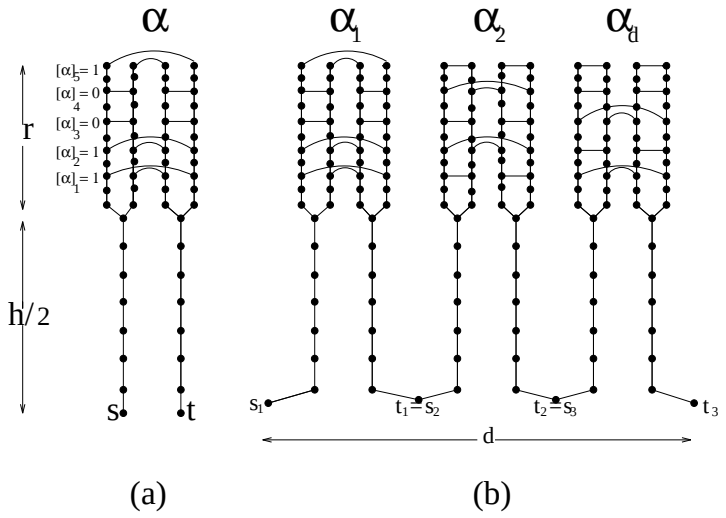

Figure 3.2: A fly swatter graph and a line of fly swatters

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.

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.

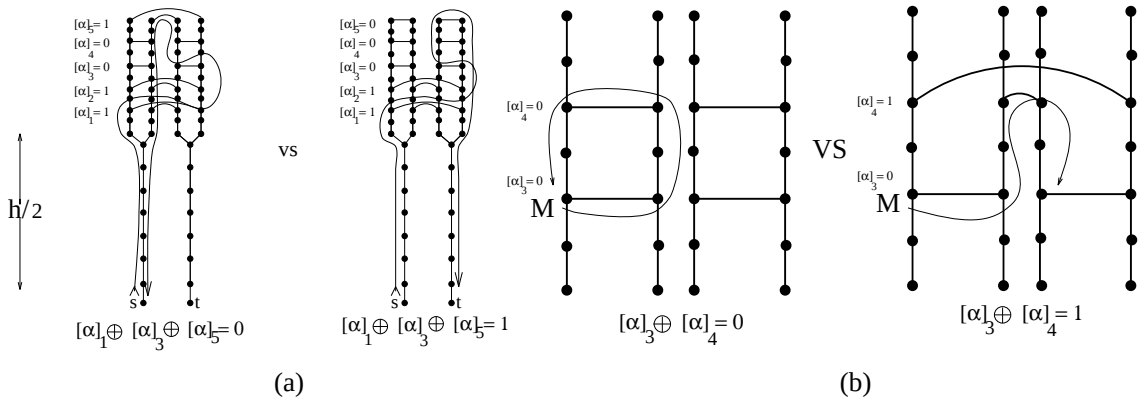

Figure 3.3: A parity question without and with a marker

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 3.3 (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 helper-parity game with the number of game instances being $d = 1$ and the amount of helper bits being $b = 0$. Recall, the input to this game consists of a non-zero vector $\alpha \in \{0,1\}^r$. The player 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. [BBRRT90] 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.

3.4 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. As mentioned in Section 3.3, 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 3.2 (b). The similarities between the parity game and the traversal of a line of fly swatter graphs should be clear, at least informally. Theorem 6 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 $(1.5 - \epsilon)d$.

The time to traverse the line of fly swatters is the product of the number of parity questions asked and the number of time steps per question. This, however, might be complicated if the JAG utilizes a marker for some of the questions and not for others. This is handled in the formal proof. However, it is instructive to consider the situation where this does not happen. Informally, we can conclude that 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 $(1.5 - \epsilon)d$ edges are traversed.

Because the input graph is built recursively on the edges of this graph, it is convenient to compare this time bound to the number of edges in the line of fly swatters instead of the number of nodes. Without a marker, the time required is roughly a factor of $(1.5 - \epsilon)$ larger than the number of edges. With a marker, it is roughly a factor of $\frac{(1.5-\epsilon)}{h}$ less than the number of edges. This difference is not very impressive, but its effect is magnified in a recursive construction.

3.5 The Recursive Fly Swatter Graphs

Let $G(\vec{\alpha}_l)$ denote the line of fly swatters specified by the vector of vectors $\vec{\alpha}_l = \left\langle \alpha_{\langle l,1 \rangle}, \ldots, \alpha_{\langle l,d \rangle} \right\rangle \in (\{0,1\}^r - \{0^r\})^d$. 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.

A crucial and difficult 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

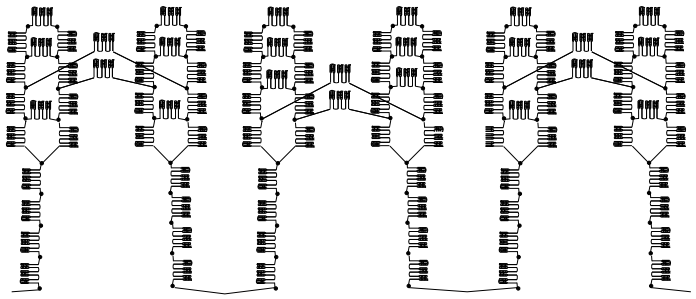

Figure 3.4: The recursive line of fly swatters graph (Sorry, the cycles in the figure have length 4 instead of 6.)

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 3.3 (b) and 3.5.

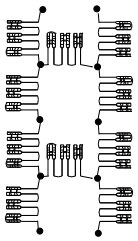

Figure 3.5: Ten super edges of a two level recursive fly swatter graph

An intuitive explanation of the lower bound can now be given. (A formal proof is provided in Section 3.8.) 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,

$$\begin{aligned} T &\geq [h(1.5-\epsilon)d]^{L-p+1}[(1.5-\epsilon)d]^{p-1} \\ &= (1.5-\epsilon)^L \times (hd)^L \times h^{-p+1} \end{aligned} \tag{3.1}$$

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 $2^{\Omega\left(\frac{\log n}{\log\log n}\right)}$, 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 gives the result in Theorem 7.

3.6 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 $p - 2 + 1 = p - 1$ 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

$$\begin{aligned} w\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right] &= \min_{\langle Q, \Pi \rangle} \log\left(h^{p\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle, Q, \Pi\right]} T\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle, Q, \Pi\right]\right) \text{ (and let)} \\ W_{[l]} &= \mathrm{Exp}_{\left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle} w\left[l, \left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle\right]. \end{aligned}$$

The motivation for using $h^{p_{[l]}} T_{[l]}$ comes from Equation 3.1. 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. Technically, it is convenient to use the logarithm of this quantity. This converts the total complexity from being the product of the complexities at each level into being the sum. We are then able to use the fact that the expectation of the sum is the sum of the expectations. 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. Finally, $W_{[l]}$ is the average cost for a uniformly chosen input $\left\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \right\rangle$. Substituting the estimate Equation 3.1 into the definition of $W_{[l]}$ gives the informal estimate of the average cost to be $\log\left(\left[(1.5 - \epsilon) hd\right]^l h\right)$ $= \left[\log h + \log d + \log(1.5 - \epsilon)\right] \times l + \log h$. This motivates Lemma 5.

3.7 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. Recall that the cost of a traversal is proportional to the logarithm of time for the traversal and hence differences in cost correspond to ratios of time.

Lemma 5 *Given an algorithm for st-traversal whose average cost to traverse the graph at the $l-1^{st}$ and the l^{th} levels are $W[l-1]$ and $W[l]$, we can produce a protocol for the helper parity game for which the average number of questions asked per game instance is*

$$Exp_{\vec{\alpha}_l} c_{\vec{\alpha}_l} \quad \leq \quad W[l] - W[l-1] - \log(h) - \log d + 1 + \frac{9p}{d}.$$

Proof of Lemma 5: Consider a fixed algorithm for st-traversal whose average cost to traverse the graph at the $l-1^{st}$ and the l^{th} levels are $W[l-1]$ and $W[l]$. 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 $\vec{\beta}$. Since

$$\begin{aligned} W[l] - W[l-1] &= \mathrm{Exp}_{\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \rangle} w\left[l, \langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \rangle\right] - \mathrm{Exp}_{\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \rangle} w\left[l-1, \langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \rangle\right] \\ &= \mathrm{Exp}_{\langle \vec{\gamma}, \vec{\beta} \rangle} \mathrm{Exp}_{\vec{\alpha}_l} \left[w\left[l, \langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \rangle\right] - w\left[l-1, \langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \rangle\right] \right], \end{aligned}$$

There exists $\vec{\gamma}'$ and $\vec{\beta}'$ such that

$$\mathrm{Exp}_{\vec{\alpha}_l} \left[w\left[l, \langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \rangle\right] - w\left[l-1, \langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \rangle\right] \right] \quad \leq \quad W[l] - W[l-1]$$

Fix vectors $\vec{\gamma}'$ and $\vec{\beta}'$ satisfying this property. These vectors 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 $\left\langle Q_{\langle l,\vec{\alpha}_l \rangle}, \Pi_{\langle l,\vec{\alpha}_l \rangle} \right\rangle$, where this 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 3.1, 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 3.1. By Claim 2, $|S_0 \cup S_1| \leq 2p+1$. 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, I 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. I 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. Because making these events occur is costly for the JAG, the JAG cannot make them occur too many times while keeping the cost $W[l] - W[l-1]$ small. Hence, the game-player in the defined protocol does not ask a lot of questions.

Before continuing, recall that Section 3.3 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 of the fly swatter 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 a 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. I 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 I 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 are 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 $s_{\langle l,i \rangle}$. By traversing a super edge, while all the pebbles contained in this entire line of fly swatters $G(\vec{\gamma}', \vec{\alpha}_l)$ is contained in this super edge, two pebbles can move on opposite sides of a super edge. They can get three full super edges between them by them respectively traversing a super edge forward and backwards. This requires traversing a super edge of the fly swatter 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 while there is a pebble contained in this copy of $G(\vec{\gamma}', \vec{\alpha}_l)$, but not contained in the super edge. These two traversals occur during disjoint intervals in time and hence the first event occurs. Because the first event does not occur, the pebbles in the fly swatter never 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.

I 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 bound the average number of questions asked by the game-player. To do this, we need to prove that the JAG requires a lot of time to make one of the two events occur and hence does not have enough time to make them occur often. 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 $G(\vec{\gamma}', \vec{\alpha}_l)$. There are two factors that effect this time: the number of pebbles used and the time to traverse a super edge $G(\vec{\gamma}')$. The game-player runs the JAG until some copy of $G(\vec{\gamma}', \vec{\alpha}_l)$ is traversed. Hence, the number of pebbles used to make the event in question happen is bounded by the number of pebbles "used" during this traversal. Recall that by definition $p\left[l, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle, Q_{\langle l,\vec{\alpha}_l \rangle}, \Pi_{\langle l,\vec{\alpha}_l \rangle}\right]$ is this number, where $\left\langle Q_{\langle l,\vec{\alpha}_l \rangle}, \Pi_{\langle l,\vec{\alpha}_l \rangle} \right\rangle$ 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. This will be abbreviated by $p_{[l,\vec{\alpha}_l]}$. The minimum cost to traverse a super edge $G(\vec{\gamma}')$ when the entire graph is

$G\left(\vec{\gamma}', \vec{\alpha}_l, \vec{\beta}'\right)$ is defined to be $w\left[l-1, \left\langle \vec{\gamma}', \vec{\alpha}_l, \vec{\beta}' \right\rangle\right]$. Abbreviate this with $w[l-1, \vec{\alpha}_l]$.

Claim 3 *For each* $i \in [1..d] - (S_0 \cup S_1)$,

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

Proof of Claim 3:
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 traversal occurs 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

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

This is at least

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

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

$$T[l-1, \vec{\alpha}_l] \quad \geq \quad 2^{w[l-1, \vec{\alpha}_l]} h^{-p\left[l-1, \vec{\alpha}_l\right]}.$$

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 $G(\vec{\gamma}')$ 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 $T[l-1,\vec{\alpha}_l] \geq 2^{w[l-1,\vec{\alpha}_l]}h^{-p\left[l,\vec{\alpha}_l\right]+1}$. 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, $2 \times 2^{w[l-1,\vec{\alpha}_l]}h^{-p\left[l,\vec{\alpha}_l\right]+1}$ 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 $T[l-1,\vec{\alpha}_l] \geq 2^{w[l-1,\vec{\alpha}_l]}h^{-p\left[l,\vec{\alpha}_l\right]}$. Therefore, the time to traverse h super edges is at least $2^{w[l-1,\vec{\alpha}_l]}h^{-p\left[l,\vec{\alpha}_l\right]+1}$.

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 $2 \times 2^{w[l-1,\vec{\alpha}_l]}h^{p\left[l,\vec{\alpha}_l\right]+1}$ time steps. ■

The final goal is to use the fact that $W[l]-W[l-1]$ for the st-connectivity algorithm is small in order to bound the average number of questions asked per game instance. Recall that, by definition, $T\left[l,\left\langle\vec{\gamma}',\vec{\alpha}_l,\vec{\beta}'\right\rangle,Q_{\langle l,\vec{\alpha}_l\rangle},\Pi_{\langle l,\vec{\alpha}_l\rangle}\right]$ is the total number of time steps occurring within the first copy of $G(\vec{\gamma}',\vec{\alpha}_l)$ to be traversed and that $p\left[l,\left\langle\vec{\gamma}',\vec{\alpha}_l,\vec{\beta}'\right\rangle,Q_{\langle l,\vec{\alpha}_l\rangle},\Pi_{\langle l,\vec{\alpha}_l\rangle}\right]$ is the number of pebbles "used" during this traversal. (Here $\left\langle Q_{\langle l,\vec{\alpha}_l\rangle},\Pi_{\langle l,\vec{\alpha}_l\rangle}\right\rangle$ specify 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.) The cost of the traversal of this $G(\vec{\gamma}',\vec{\alpha}_l)$ is defined to be $w\left[l,\left\langle\vec{\gamma}',\vec{\alpha}_l,\vec{\beta}'\right\rangle\right] = \min_{\langle Q,\Pi\rangle}\log\left(h^{p\left[l,\left\langle\vec{\gamma}',\vec{\alpha}_l,\vec{\beta}'\right\rangle,Q,\Pi\right]}T\left[l,\left\langle\vec{\gamma}',\vec{\alpha}_l,\vec{\beta}'\right\rangle,Q,\Pi\right]\right)$. This minimum is no more than that for the $\left\langle Q_{\langle l,\vec{\alpha}_l\rangle},\Pi_{\langle l,\vec{\alpha}_l\rangle}\right\rangle$ chosen by the JAG-Helper. (Without loss of generality, we can assume that the same $\left\langle Q_{\langle l,\vec{\alpha}_l\rangle},\Pi_{\langle l,\vec{\alpha}_l\rangle}\right\rangle$ gives the minimum.) Abbreviate these values to $T[l,\vec{\alpha}_l]$, $p[l,\vec{\alpha}_l]$ and $w[l,\vec{\alpha}_l]$. The average number of questions asked per game instance, in the helper parity game defined above, is

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

Before bounding this, let us bound the following slightly different value.

$$\mathrm{Exp}_{\vec{\alpha}_l}\log\left(c_{\vec{\alpha}_l}d-4p-2\right)$$

$$
\begin{aligned}
&\leq \mathrm{Exp}_{\vec{\alpha}_l} \log \left(-4p - 2 \ + \ 2|S_0 \cup S_1| \ + \sum_{i \in [1..d]-(S_0 \cup S_1)} \left\{ \begin{array}{ll} 1 & \text{if } \bigoplus_{j \in E_i} [\alpha_i]_j = 1 \\ 2 & \text{otherwise} \end{array} \right\} \right) \quad \text{(by Claim 3)} \\
&\leq \mathrm{Exp}_{\vec{\alpha}_l} \log \left(\sum_{i \in [1..d]-(S_0 \cup S_1)} 2^{-w[l-1,\vec{\alpha}_l]} h^{p\left[l,\vec{\alpha}_l\right]-1} T[l,\vec{\alpha}_l,i] \right) \\
&\leq \mathrm{Exp}_{\vec{\alpha}_l} \log \left(2^{-w[l-1,\vec{\alpha}_l]} h^{p\left[l,\vec{\alpha}_l\right]-1} T[l,\vec{\alpha}_l] \right) \\
&\quad \text{(by choice of } \left\langle Q_{\langle l,\vec{\alpha}_l \rangle}, \Pi_{\langle l,\vec{\alpha}_l \rangle} \right\rangle, \ w[l,\vec{\alpha}_l] = \log \left(h^{p\left[l,\vec{\alpha}_l\right]} T[l,\vec{\alpha}_l] \right)) \\
&= \mathrm{Exp}_{\vec{\alpha}_l} \log \left(2^{-w[l-1,\vec{\alpha}_l]} h^{p\left[l,\vec{\alpha}_l\right]-1} \ \times \ 2^{w[l,\vec{\alpha}_l]} h^{-p\left[l,\vec{\alpha}_l\right]} \right) \\
&= \mathrm{Exp}_{\vec{\alpha}_l} \left(w[l,\vec{\alpha}_l] - w[l-1,\vec{\alpha}_l] - \log(h) \right) \\
&= W[l] - W[l-1] - \log(h)
\end{aligned}
$$

In order to handle the $-4p-2$, note that $\log(c_{\vec{\alpha}_l} d - 4p - 2) = \log(c_{\vec{\alpha}_l} d) + \log\left(1 - \frac{4p+2}{c_{\vec{\alpha}_l} d}\right)$ $\geq \log(c_{\vec{\alpha}_l} d) - 2\frac{4p+2}{c_{\vec{\alpha}_l} d} \geq \log(c_{\vec{\alpha}_l} d) - \frac{9p}{d}$, because $1 - x \geq 4^{-x}$ if $x \in [0, \frac{1}{2}]$. Handling the other logarithm is more difficult, because in general $\mathrm{Exp}\left[\log(c_{\vec{\alpha}_l} d)\right) \leq \log\left(\mathrm{Exp}\left[c_{\vec{\alpha}_l} d\right]\right)$. In fact, the left hand side can be made arbitrarily small by making the variance of the $c_{\vec{\alpha}_l}$ values large. Luckily, this diversity is limited by $1 \leq c_{\vec{\alpha}_l} \leq 2$, because the game never charges less than 1 question per game instance or more than 2. Suppose that ϵ is such that $\mathrm{Exp}_{\vec{\alpha}_l} c_{\vec{\alpha}_l} = 1.5 - \epsilon$. Then the worst case diversity is when a $\left(\frac{1}{2} + \epsilon\right)$ fraction of the $\vec{\alpha}_l$ values give $c_{\vec{\alpha}_l} = 1$ and a $\left(\frac{1}{2} - \epsilon\right)$ fraction give $c_{\vec{\alpha}_l} = 2$. This gives

$$
\begin{aligned}
\mathrm{Exp}_{\vec{\alpha}_l} \log(c_{\vec{\alpha}_l} d) &\geq \left(\frac{1}{2} + \epsilon\right) \log(1d) + \left(\frac{1}{2} - \epsilon\right) \log(2d) \ = \ \log d + \left(\frac{1}{2} - \epsilon\right) \\
&= \log d + \frac{1}{2} - \left(1.5 - \mathrm{Exp}_{\vec{\alpha}_l} c_{\vec{\alpha}_l}\right) \\
&= \mathrm{Exp}_{\vec{\alpha}_l} c_{\vec{\alpha}_l} + \log d - 1.
\end{aligned}
$$

In conclusion,

$$
\begin{aligned}
\mathrm{Exp}_{\vec{\alpha}_l} c_{\vec{\alpha}_l} &\leq \mathrm{Exp}_{\vec{\alpha}_l} \log(c_{\vec{\alpha}_l} d - 4p - 1) - \log d + 1 + \frac{9p}{d} \\
&\leq W[l] - W[l-1] - \log(h) - \log d + 1 + \frac{9p}{d}. \ \blacksquare
\end{aligned}
$$

3.8 The Formal Proof

The final step is to prove the theorem.

Theorem 7′ *For every constant $z \geq 2$, the average time for st-traversal by helper JAG with $p \leq \frac{1}{28z} \frac{\log n}{\log\log n}$ pebbles and $q \leq 2^{\log^z n}$ states is at least $n \times 2^{\frac{1}{25z} \frac{\log n}{\log\log n}}$, where the input graph $G(\vec{\alpha}_1, ..., \vec{\alpha}_L)$ is chosen by uniformly choosing $\vec{\alpha}_1, ..., \vec{\alpha}_L$ from $(\{0,1\}^r - \{0^r\})^d$ (for most sufficiently large n).*

Proof of Theorem 7′: Fix any constant $z \geq 2$ and suppose $p \leq \frac{1}{28z} \frac{\log n}{\log\log n}$ and $q \leq 2^{\log^z n}$. Set $\epsilon = 0.0125$, $\delta = 0.0002$, $r = 13$, and $d = \lceil \frac{2}{\delta} \log^z n \rceil \geq \frac{1}{\delta}(\log^z n + p \log n) \geq \frac{1}{\delta} \log(qn^p) = \frac{1}{\delta} b$. Then,

by Theorem 6, $c \geq 1.5 - \epsilon$. As well $p \in o(d)$. Therefore, by Lemma 5,

$$W_{[l]} \geq W_{[l-1]} + \log h + \log d + \left(\frac{1}{2} - \epsilon - o(1)\right).$$

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 gives $W_{[0]} \geq \mathrm{Exp}_{\langle \vec{\gamma}, \vec{\alpha}_l, \vec{\beta} \rangle} \min_{\langle Q, \Pi \rangle} [p_{[0]} \log h + \log T_{[0]}] \geq \log h$ and hence $W_{[L]} \geq \left[\log h + \log d + \left(\frac{1}{2} - \epsilon - o(1)\right)\right] L + \log h$. By definition,

$$W_{[L]} = \mathrm{Exp}_{\langle \vec{\gamma}, \vec{\alpha}_L, \emptyset \rangle} \min_{\langle Q, \Pi \rangle} \log\left(h^{p_{[L]}} T_{[L]}\right) \leq p \log h + \mathrm{Exp}_{\langle \vec{\gamma}, \vec{\alpha}_L, \emptyset \rangle} \min_{\langle Q, \Pi \rangle} \log\left(T_{[L]}\right)$$

Rearranging this equation gives the average time for the helper JAG to traverse the entire input graph.

$$\begin{aligned}
&\mathrm{Exp}_{\langle \vec{\gamma}, \vec{\alpha}_L, \emptyset \rangle} \min_{\langle Q, \Pi \rangle} T_{[L]} \\
&= 2^{\log\left(\mathrm{Exp}\ \min\ T_{[L]}\right)} \text{ (which by concavity of the log function)} \\
&\geq 2^{\mathrm{Exp}\ \log\left(\min\ T_{[L]}\right)} \geq 2^{W_{[L]} - p \log h} \geq 2^{\left(\frac{1}{2} - \epsilon - o(1)\right) L} (hd)^L h^{-p+1}.
\end{aligned}$$

Set $h = 1922$ and $\epsilon' = \frac{6r}{h} = \frac{39}{961}$. Then, the number of vertices in $G(\vec{\alpha}_1, ..., \vec{\alpha}_L)$ is $n \leq [(h + 6r)\, d]^L \leq [(1 + \epsilon')\, hd]^L$ and $L \geq \frac{\log n}{\log((1+\epsilon')hd)} \geq \frac{\log n}{z \log\log n + O(1)}$. This gives

$$\begin{aligned}
&\mathrm{Exp}_{\langle \vec{\gamma}, \vec{\alpha}_L, \emptyset \rangle} \min_{\langle Q, \Pi \rangle} T_{[L]} \\
&\geq 2^{\left(\frac{1}{2} - \epsilon - o(1)\right) L} \times (hd)^L \times h^{-p+1} \\
&= \left[(1 + \epsilon')\, hd\right]^L \times 2^{\left(\frac{1}{2} - \epsilon - o(1) - \log(1+\epsilon')\right) L} \times 2^{-p \log h + 1} \\
&\geq [n] \times 2^{\left(\frac{1}{2} - 0.0125 - o(1) - \log(1+0.0406)\right)\left[\frac{\log n}{z \log\log n + O(1)}\right]} \times 2^{-\left[\frac{1}{28z} \frac{\log n}{\log\log n}\right] \log 1922} \\
&\geq n \times 2^{\frac{1}{25z} \frac{\log n}{\log\log n}} \quad \blacksquare
\end{aligned}$$

Chapter 4

Directed st-Connectivity on a JAG

A lower bound $T \times S^{\frac{1}{2}} \in \Omega\left(mn^{\frac{1}{2}}\right)$ on the time-space tradeoff to compute directed st-connectivity on a JAG is proved in this chapter. The proof is simple and elegant and applies even when the number of states is not counted as part of the space.

4.1 Comb Graphs

The lower bound on the time-space tradeoff is proved for a subdomain of acyclic directed graphs, referred to as the family of **comb graphs**. A comb graph, illustrated in Figure 4.1, is composed of a **back**, χ **teeth**, plus the distinguished node t. The back of the comb consists of a directed path of n nodes $v_1, \ldots, v_n$. The first node v_1 is the distinguished node s. The r^{th} tooth consists of the directed path $u_{\langle r,1\rangle}, \ldots, u_{\langle r,l\rangle}$. The length of each tooth will be $l = \frac{n}{\chi}$ so that the total number of nodes is $N = 2n + 1$.

There are m ($\geq n$) directed **connecting edges** $e_1, \ldots, e_m$ each going from a back node v_i to the top of one of the teeth in such a way that that the out-degree of any two back nodes can differ by at most 1. In particular, if $m = n$, the out-degree of the graph is two. More formally, for $j \in [1..m]$, the connecting edge edge e_j is the $\lceil \frac{j}{n} \rceil^{th}$ edge emanating from back node $v_{\langle (j \bmod n)+1\rangle}$ and has label $\lceil \frac{j}{n} \rceil$. The variables $y_1, \ldots, y_m \in [1..\chi]$ will be used to specify which tooth each of the connecting edges leads to. Specifically, $y_j = r$ means that the edge e_j leads to the top node of the r^{th} tooth. We will allow double edges, so it is of no concern if two edges from the same back node v_i go to the same tooth.

If there is to be a directed path from s to t then the node t is attached to the bottom of at least one of the teeth. The variables $\alpha_1, \ldots, \alpha_\chi \in \{0, 1\}$ will be used to specify for each of the teeth whether this is the case. Specifically, $\alpha_r = 1$ if the bottom node of the r^{th} tooth has a directed edge to node t. If $\alpha_r = 0$, then there is a self loop edge from the bottom node to itself in order to keep the degree fixed.

The number of teeth χ is a parameter that will chosen after the space allocated to the model is fixed. In Theorem 8, it is set to $n^{\frac{1}{2}}p^{\frac{1}{2}} >> p$ so that there are more teeth than pebbles. In Theorem 9, it is set to $m^{\frac{1}{3}}n^{\frac{1}{3}}S^{\frac{1}{3}} >> S$ so that the NNJAG is unable to store even a bit of information about each tooth.

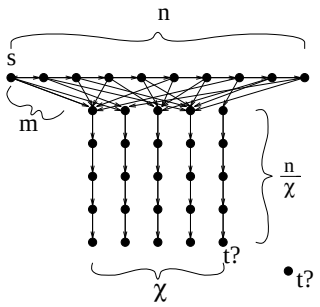

Figure 4.1: A comb graph

Intuitively, solving *st*-connectivity for comb graphs is difficult because each tooth needs to be traversed to the bottom before the JAG can be sure that t is not connected to any of them. However, because the amount of space is limited, it is difficult for the JAG to "remember" which of the teeth have been traversed already. Therefore, some teeth may get traversed many times before the JAG is sure that they all have been traversed.

4.2 JAGs with Many States

There are two basic approaches to computing *st*-connectivity on comb graphs. One is the brute force approach. For each connecting edge e_i, the JAG walks a pebble to the bottom of the tooth attached to it. This requires $m \times l$ time steps and two pebbles.

Another approach is to learn enough about which connecting edges are attached to the same teeth, so that no tooth needs to be traversed more than once. This is done by choosing two connecting edges e_i and e_j and moving a pebble to the top of the tooth attached to e_i and another pebble to the top of the tooth attached to e_j. The edges e_i and e_j are attached to the same tooth if and only if the pebbles collide. This approach requires $\Theta\left(\frac{\chi m}{p}\right)$ time steps. This is proved by reducing the problem to the following game.

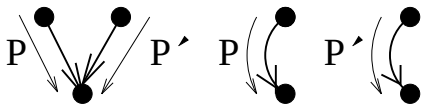

Figure 4.2: The connecting edges are connected to the same tooth or different teeth

The game is parameterized by m, χ, and μ and the input consists of a partition of the edges $e_1, \ldots, e_m$ into χ non-empty parts. The player is able to specify two edges and query whether or not they are in the same part. The game is over when the player has determined a set S of μ edges that cover each of the χ parts of the input partition, i.e. for each of the χ parts, there is an edge from the part that is included in S.

Lemma 6 *The partition game requires at least* $\frac{1}{2}(\chi - 1)(m - \mu)$ *queries.*

An upper bound of χm is easy for completely determining the partition. Simply, query for each edge e_i whether or not e_1 and e_i are in the same part. From these m queries, you completely learn the contents of the part containing e_1. Delete these edges and repeat once for each part.

Proof of Lemma 6 [Im93]: For the purpose of this proof, $e_1, \ldots, e_m$ will be referred to as nodes instead of edges, because edges $\{e_i, e_j\}$ are required for the proof. The adversary, for the proof, maintains disjoint parts $P_1, \ldots, P_{\chi-1} \subseteq \{e_1, .., e_m\}$ and an undirected graph H. The adversary adds a node to the part P_r when he fixes the node to be within the r^{th} part of the input partition and he adds an edge $\{e_i, e_j\}$ to H when he reveals to the player that these nodes are in different parts. A property of H that is maintained is that the degree of every node not in $\cup_{r\in[1..\chi-1]}P_r$ is at most $\chi - 2$.

When the player asks a question $\{e_i, e_j\}$ for the first time, the adversary does the following. For each of e_i and e_j, if it is not in $\cup_{r\in[1..\chi-1]}P_r$ and has degree $\chi-2$, then it is added to one of the parts P_r that contains none of e_i's (e_j's) neighbors in H. There are $\chi - 1$ parts, so by the pigeon hole principle such a part exists. If e_i and e_j are both added to parts, it does not matter whether or not they go into the same part. Now the adversary responds to the question. If e_i and e_j are contained in the same part P_r, the adversary reveals this information. Otherwise, the adversary answers that e_i and e_j are in different parts and adds the edge $\{e_i, e_j\}$ to H.

After only $\frac{1}{2}(\chi-1)(m-\mu) - 1$ queries, there are at least $\mu + 1$ nodes not contained in $\cup_{r\in[1..\chi-1]}P_r$. This is because a node is not added to a part until $\chi-1$ questions are asked about it. However, a single query asks about two nodes. Therefore, $\frac{1}{2}(\chi-1)(m-\mu)$ queries are required for all but μ of the m nodes to be added. Hence, with one fewer query, there are at least $\mu + 1$ nodes that have not been added to a part.

At the end of the computation, the player must specify a set S of μ nodes that cover each of the χ parts of the input partition. By the pigeon hole principle, there must be a node e_* that is amongst the $\mu+1$ nodes not contained in $\cup_{r\in[1..\chi-1]}P_r$ and is not amongst the μ nodes in S specified by the player. The adversary then fixes the χ^{th} part P_χ to be the singleton part containing only e_*. Each of the nodes that has not yet been added to a part is added to one of the first $\chi-1$ parts that contains none of its neighbors in H. This defines a partition $P_1, \ldots, P_\chi$ which is consistent with all the answers given by the adversary. Hence, the player's solution to the computation must be correct, however, it is not. The part P_χ is not covered by the players set of nodes S, since e_* is not in it. ■.

Theorem 8 *All JAGs with p pebbles (even with an arbitrarily large number of states) that compute directed st-connectivity require time $\Omega\left(mn^{\frac{1}{2}}/p^{\frac{1}{2}}\right)$.*

Proof of Theorem 8: We will show that $T \geq \text{Min}\left(\frac{m}{2} \times l, \frac{\frac{1}{2}(\chi-1)(\frac{m}{2})}{(p-1)}\right)$. If a JAG has p pebbles, then setting χ to $n^{\frac{1}{2}}p^{\frac{1}{2}}$ gives the required bound $T \in \Omega\left(mn^{\frac{1}{2}}/p^{\frac{1}{2}}\right)$, since $l = \frac{n}{\chi}$.

The proof reduces the *st*-connectivity problem to the above partition game with the parameters m, χ, and $\mu = \frac{m}{2}$. Suppose by way of contradiction, that there is a JAG algorithm that uses less than this amount of time. From this, a protocol for the partition game is constructed that beats the bound given in Lemma 6. The input to the game is a partition of $e_1, \ldots, e_m$ into χ parts $P_1, \ldots, P_\chi$. The comb graphs corresponding to this partition are those in which, for each $r \in [1..\chi]$, the connecting edges in P_r are connected to the r^{th} tooth. The comb graphs considered by the game-player are only those that are not connected from s to t. In particular, fix $\alpha_r = 0$ for all $r \in [1..\chi]$. Hence, a partition completely specifies a comb graph.

The game-player simulates the running of the JAG. As the JAG computation proceeds, he

associates each pebble that is contained within a tooth, with the connecting edge e_i through which it entered the tooth. If the pebble had jumped into the tooth, then it is associated the connecting edge e_i that the pebble jumped to was associated with. Whenever there is a pebble associated with the connecting edge e_i and another pebble associated with e_j at the same time, the JAG might learn whether or not e_i and e_j are connected to the same tooth. When this first happens, the game-player "queries $\{e_i, e_j\}$" and learns whether or not they are in the same part. Similarly, whenever there is a pebble associated with the connecting edge e_i that traverses down to the bottom of the tooth connected to e_i, the JAG might learn whether or not this tooth is connected to t. When this happens, the game-player adds e_i to the set S that is supposed to cover the χ parts (teeth).

The game-player is able to continue this simulation because the JAG computation is the same for every comb graph consistent with the information revealed by the queries. This follows from the following two observations. First, although the in-degree of the nodes on the tops of the teeth depend on the particular comb graph, the JAG has no access to the in-degree of nodes. Secondly, when two pebbles enter teeth via two different connecting edges e_i and e_j, the answers to the queries ensure that they are either in the same tooth for each input graph or in different teeth for each graph. Thus if two pebbles meet within the computation for one of the input graphs still being considered, then they meet within the computation for each of the graphs.

During the entire computation, the game-player "queries" at most $\frac{1}{2}(\chi-1)\left(\frac{m}{2}\right)-1$ different pairs $\{e_i, e_j\}$. This is because during one time step of the JAG computation only one pebble is allowed to move. This time step causes the game-player to make a query only if this pebble moves into the tooth attached to say e_i, while there is another pebble already in the tooth attached to e_j. There are only $p-1$ other pebbles. Therefore, this time step cause the game-player to make at most $p-1$ queries. Therefore, in $\frac{\frac{1}{2}(\chi-1)\left(\frac{m}{2}\right)}{(p-1)}-1$ time steps, at most $\frac{1}{2}(\chi-1)\left(\frac{m}{2}\right)-1$ queries can be made.

Note as well, the game-player adds at most $\frac{m}{2}$ connecting edges to the set S. The reason is that the number of JAG computation steps required to move a pebble into a tooth via a connecting edge and then to the bottom of the tooth is at least l, the length of the tooth. The JAG computation proceeds for less than $\left(\frac{m}{2}\right)l$ time steps. Therefore, this is done for at most $\frac{m}{2}$ connecting edges.

Finally, for every game input $P_1, \ldots, P_\chi$, the set S formed from this game protocol must cover all χ of the parts. By way of contradiction, suppose that there is a partition $P_1, \ldots, P_\chi$ and an r for which S is disjoint from the part P_r. This partition corresponds to a comb graph, which we will denote by G. Never, during the computation on G, does a pebble traverses to the bottom of the r^{th} tooth. If a pebble did, then the pebble would need to enter the tooth via a connecting edge. This connecting edge would need to be contained in both P_r and S. It follows that a pebble never is on the bottom node $u_{\langle r,l\rangle}$ of this tooth.

Let G' be the same graph except that the node t is attached to the bottom of the r^{th} tooth, i.e. $\alpha_r = 1$. The JAG computation is identical on the graphs G and G', because the JAG must have a pebble on node $u_{\langle r,l\rangle}$ in order to know whether or not there is an out-going edge from it to t. Pebbles located on node t do not give the JAG any information about incoming edges. In fact, because t has no outgoing edges, pebbles on t can only move by jumping.

Because the computation is the same on G and G', the JAG must give an incorrect answer for one of the graphs. This is a contradiction, if the JAG algorithm must always give the correct answer. Hence, the set S produced by the game protocol must cover all the parts of the game's input. ■

Chapter 5

Directed st-Connectivity on a NNJAG

In this chapter, I prove another lower bound on the time-space tradeoff for computing directed st-connectivity on comb graphs. The bound is a little weaker, but it applies to the stronger NNJAG model and its probabilistic version with either one-sided or two-sided error.

My time-space tradeoff for st-connectivity is proved for the NNJAG by translating any efficient NNJAG algorithm into a (r-way) branching program. Defining a measure of progress for a decision problem on such a general model of computation still remains elusive. However, for this result, the branching program inherits enough structure from the NNJAG that an effective measure of progress can be defined.

The lower bound follows the framework introduced in [BFKLT81] and used in [BC82] (See Section 1.2.) If the computation time is short, then for each input there must be a short sub-branching program in which lots of the "progress" required for the input is made. However, no sub-branching program is able to accomplish this for many inputs. Therefore, in order to handle all of the inputs, the branching program must be composed of many of these sub-branching programs. This means that the branching program has lots of nodes and hence uses lots of "space".

5.1 A Probabilistic NNJAG with One Sided Error

A probabilistic algorithm is said to allow **one-sided error** for a language L if for every input not in L, the correct answer is given, but for inputs in L, the incorrect answer may be given with some bounded probability. We will be considering algorithms with one-sided error for non-st-connectivity. Specifically, the algorithms must answer correctly, when there is a directed path from s to t. The result is strengthened by considering a random input chosen from a natural input distribution $\mathcal{D}$ on graphs for which s and t are not connected. (Let st-conn denote the set of graphs that are connected.) The result is stated as follows.

Theorem 9 *There exists a probability distribution $\mathcal{D}$ on directed graphs of out-degree two that are not in st-conn such that for every probabilistic NNJAG solving directed non-st-connectivity with*

one-sided error

$$\Pr_{G\in\mathcal{D},R\in\{0,1\}^*}\left[T_{\langle G,R\rangle} \leq 0.09\frac{m^{\frac{2}{3}}n^{\frac{2}{3}}}{S^{\frac{1}{3}}} \quad \textit{and} \quad \langle G,R\rangle \in Corr\right] \leq 2^{-S},$$

where $T_{\langle G,R\rangle}$ is the computation time for input G and random bit string $R \in \{0,1\}^*$ and define $Corr$ is the set of $\langle G,R\rangle$ for which the correct answer to st-connectivity is given.

This formulation of the bound is different than that used by Yao [Ya77]. However, his formulation follows as a simple corollary. When considering only algorithms for which the probability of error is no more than some constant λ, the expected running time is at least $\left(1-\lambda+2^{-S}\right)T_{max}$, where T_{max} is the bound given above. However, I consider my formulation to be more interesting, because it considers the running time only for those $\langle G,R\rangle$ for which the correct answer is given. The following upper bound demonstrates that the running time when the incorrect input is given is of no interest. Suppose that there is a deterministic algorithm whose worst case time is T'_{max}. The following is a probabilistic algorithm that allows only one-side error, has error probability λ, and the expected running time is $(1-\lambda)T'_{max}$. On each input, flip a coin. With probability $1-\lambda$, compute the correct answer using T'_{max} time steps. Otherwise, answer that the input is in *st*-conn. Note that the bound $\left(1-\lambda+2^{-S}\right)T_{max}$ is fine when $\lambda \leq \frac{1}{2}-\epsilon$. However, when the algorithms allow only one-sided error, it is interesting to consider algorithms that give the correct answer with only very small probability, for example $1-\lambda = 2\times 2^{-S}$. It may be difficult to quickly obtain even this many correct answers while assuring to give the correct answer when the input is in *st*-conn. In this case, Yao's formulation gives a very poor result, namely that the expected time is at least $2^{-S}T_{max}$. However, the formulation in the theorem is still interesting.

The proof of Theorem 9 does not consider probabilistic NNJAGs, but deterministic NNJAG on a random input. Recall that Yao [Ya77] proves that a lower bound on the average time on a deterministic algorithm gives an expected case lower bound for a random algorithm. However, there is a problem when the errors are allowed. If the lower bound on the average time on a deterministic algorithm applies when the algorithm allows errors with 2λ probability, then Yao only gives the expected case lower bound on for random algorithm directly applies when the algorithm allows errors with only λ probability. This factor of 2 in the error probability is significant. For two-sided error, an algorithm can get the correct answer with probability $\frac{1}{2}$, simply by guessing and hence the strongest result possible will apply to algorithms with error probability $\lambda = \frac{1}{2}-\epsilon$. Hence, the strongest result given by Yao's reduction is for $\lambda = \frac{1}{4}-\frac{\epsilon}{2}$. This is a weaker result. Using the formulation in Theorem 9, this problem does not arise. The proof proves that for any deterministic algorithm

$$\Pr_{G\in\mathcal{D}}\left[T_G \leq 0.09\frac{m^{\frac{2}{3}}n^{\frac{2}{3}}}{S^{\frac{1}{3}}} \text{ and } G \in Corr\right] \leq 2^{-S}.$$

It follows that the same is true for a randomly chosen algorithm.

This chapter is structured as follows. Section 5.2 defines the probability distribution on the input comb graphs. Section 5.3 defines a measure of progress for a computation. Section 5.4 describes how to convert a NNJAG protocol into a branching program. Section 5.5 provides the basic framework of the proof. Section 5.6 does the same for the two-sided error result. Section 5.7 states the required Chernoff results. Finally, Section 5.8 proves the technical probabilistic lemma that completes the proof.

5.2 The Probability Distribution $\mathcal{D}$ on Comb Graphs

The input domain consists of the same comb graphs as used in Theorem 8. The only difference is that the model requires that the input includes a "name" for each of the nodes. These names could be assigned arbitrarily. However, we will simply use the names v_i and $u_{\langle r,j\rangle}$ that were used to describe the graph. The effect is that the model always knows which node within the comb graph structure each pebble is on. Considering this fixed naming only strengths the lower bound result.

The probability distribution $\mathcal{D}$ is defined by constructing comb graphs as follows. Set $\alpha_r = 0$ for all $r \in [1..\chi]$. Thus all inputs $G \in \mathcal{D}$ are not in st-conn. What remains is to set the random variables $y_1, \ldots, y_m$ specifying which teeth the connecting edges $e_1, \ldots, e_m$ are attached to. Randomly partition the teeth into two equal size subsets $easyteeth_G$ and $hardteeth_G \subseteq [1..\chi]$. Randomly choose $\frac{\chi}{2}$ of the connecting edges and put the associated variables y_j in the set $hardedges_G$. Randomly attach each of these "hard" connecting edges to one of the "hard" teeth in a one-to-one way. The set $easyedges_G$ is defined to contain the remaining y_j variables. Independently assign each $y_j \in easyedges_G$ a tooth from $easyteeth_G$ chosen uniformly at random.

5.3 The Definition of Progress

The lower bound measures, for each input G and each step in the computation, how much progress has been made towards solving st-connectivity. We will say that the amount of **progress** made is the number of hard teeth, i.e. $r \in hardteeth_G$, that have had a pebble on their bottom node $u_{\langle r,l\rangle}$ at some point so far during the computation. It turns out that if the correct answer has been obtained for $G \notin st$-conn, then lots of progress must have been made.

Lemma 7 *For every comb graph $G \notin st$-conn, if $G \in Corr$, then the computation for G must make $\frac{\chi}{2}$ progress.*

Proof of Lemma 7: Suppose that $G \notin st$-conn and there is a hard tooth $r \in hardteeth_G$ such that during the computation there is never a pebble on bottom node of this tooth. Let G' be obtained from G by connecting the bottom node of the r^{th} tooth to t, i.e. set $\alpha_r = 1$. The NNJAG model is defined such that it can only learn whether there is a directed edge from $u_{\langle r,l\rangle}$ to t by having a pebble on node $u_{\langle r,l\rangle}$. Therefore, the computation on G and G' is the same. Since $G' \in st$-conn, the answer given must be that the graph is connected. Since $G \notin st$-conn, this implies that $G \notin Corr$. ■

The next lemma uses the fact that NNJAG is not a random access machine to prove that l time steps are required to make progress for one tooth.

Lemma 8 *If at some step, the r^{th} tooth does not contain a pebble, then a pebble must enter the tooth via one of the connecting edges and each edge in the tooth must be traversed by some pebble, before a pebble arrives at the bottom of this tooth.*

Proof of Lemma 8: The NNJAG model does not allow a pebble to arrive at a node unless there is another pebble to jump to or it walks there. ■

Moving a pebble to the bottom of a tooth in itself requires too little time for progress to be sufficiently costly for a superlinear lower bound. Additional cost occurs because many easy teeth

must be traversed before a hard tooth is found. The distribution $\mathcal{D}$ on comb graphs is defined so that the easy teeth are accessed by most of the connecting edges, hence are easy to find. This is why arriving at the bottom of these teeth is not considered to be progress. On the other hand, the hard teeth are each attached to only one connecting edge and hence are hard to find.

5.4 Converting an NNJAG into a Branching Program

The proof technique is to convert a NNJAG algorithm into a branching program. In general, proving lower bounds on branching programs is very difficult. However, the branching program that we will obtain will have "structure" imposed on it by the structure of the NNJAG model. Lemmas 7 and 8 characterize the required structure.

Consider any fixed NNJAG algorithm. The leveled branching program $\mathcal{P}$ is formed as follows. There is a node $\langle Q, \Pi, T \rangle$ in $\mathcal{P}$ for every configuration $\langle Q, \Pi \rangle$ of the NNJAG algorithm and time step $T \in [1..T_{max}]$, where T_{max} is the bound given in the theorem. An NNJAG configuration $\langle Q, \Pi \rangle$ is specified by the current state $Q \in [1..q]$ and the position of the p pebbles $\Pi \in [1..N]^p$. *Start, accept,* and *reject* states of the NNJAG translate to *start, accept,* and *reject* configuration nodes of the branching program, respectively. There is a directed edge from configuration node $\langle Q, \Pi, T \rangle$ to $\langle Q', \Pi', T+1 \rangle$ in $\mathcal{P}$, if there exists a comb graph for which our fixed NNJAG algorithm would cause this transition. Let us consider the possibilities.

Suppose that in configuration $\langle Q, \Pi \rangle$, our NNJAG algorithm has some pebble jump to another pebble. Π specifies the current positions of the pebbles, hence the resulting positions are uniquely defined. Similarly, the resulting state is uniquely defined, because no new information about the input graph is learned. Therefore, the configuration node $\langle Q, \Pi, T \rangle$ in $\mathcal{P}$ has out-degree one.

If, in the configuration $\langle Q, \Pi \rangle$, the NNJAG algorithm causes some pebble on a back node to walk the edge to the next back node, then the configuration node has out-degree one as well. This is also the case if there is some pebble on a tooth node which walks to the next node along the tooth.

Next, suppose there is a pebble on a back node that the NNJAG has walk the connecting edge e_j into the tooth it is attached to. The pebble will arrive on the top node of one of the teeth. Which tooth depends on the value of the variable y_j. It follows that the out-degree of such a configuration node is χ.

Finally, suppose there is a pebble on the bottom of the r^{th} tooth that the NNJAG has walk the directed edge down. If $\alpha_r = 1$, the pebble arrives at t, otherwise it remains where it is. Hence, the out-degree of this configuration node is two.

The time step T is included in the configuration $\langle Q, \Pi, T \rangle$ so that $\mathcal{P}$ is acyclic and leveled. Although the NNJAG computation may run arbitrarily long for some G, we will only be concerned about the first $T_{max} \leq n^2$ steps. The number of nodes in $\mathcal{P}$ is $q \times n^p \times n^2 \in 2^{(1+o(1))S}$, where $S = \log_2 q + p \log_2 n$ is the **space** of the NNJAG. Hence, $(1 + o(1))\, S$ is the space used by $\mathcal{P}$.

5.5 The Framework for Proving Lower Bounds on Branching Programs

The first step of the [BC82] framework is to break the leveled branching program $\mathcal{P}$ into a collection of shallow sub-branching programs. This is done by breaking $\mathcal{P}$ into layers of $h = \frac{\chi}{4}$ levels each and considering the sub-branching programs rooted at each node on the inter-layer boundaries. We now prove that for the inputs that make lots of progress in a small amount of time, there must be a sub-branching program $\widehat{\mathcal{P}}$ that makes quite a bit of progress for this input.

Lemma 9 *If $G \notin$ st-conn, $T_G \leq T_{max}$, and $G \in Corr$, then at least one of these sub-branching programs makes at least $\frac{\frac{\chi}{2}}{\frac{T_{max}}{h}}$ progress on input G.*

Proof of Lemma 9: Consider such an input G. By Lemma 7, the computation on G must make at least $\frac{\chi}{2}$ progress. Because the branching program $\mathcal{P}$ is leveled, the computation on G passes the root of only one sub-branching program $\widehat{\mathcal{P}}$ at each of the $\frac{T_{max}}{h}$ inter layer boundaries. Therefore, one of these sub-branching programs must make $\frac{\chi}{2}/\frac{T_{max}}{h}$ of the required $\frac{\chi}{2}$ progress. ■

The next step is to prove that a shallow sub-branching program cannot make this much progress for many inputs. Consider one of the sub-branching programs $\widehat{\mathcal{P}} \in \mathcal{P}$. We will determine how much progress it makes for every input $G \in \mathcal{D}$ (even if the computation on input G never reaches the root of $\widehat{\mathcal{P}}$).

Recall that each node $\langle Q, \Pi, T \rangle$ of the branching program specifies the location of every pebble in the comb graph. Define $\mathcal{F} \subseteq [1..\chi]$ to be the set of teeth that contain pebbles at the root of $\widehat{\mathcal{P}}$. Because there are only p pebbles, $|\mathcal{F}| \leq p$. For each input $G \in \mathcal{D}$, define $\mathcal{C}_G \subseteq [1..\chi]$ to be the set of teeth that do not contain pebbles at the root of $\widehat{\mathcal{P}}$, yet whose bottoms contain a pebble at some point during the computation by $\widehat{\mathcal{P}}$ on G. By Lemma 8, each edge of each tooth in $\mathcal{C}_G$ must be traversed by some pebble. The teeth have length l and the computation by $\widehat{\mathcal{P}}$ performs only h steps; therefore $|\mathcal{C}_G| \leq \frac{h}{l}$. Let $c = \frac{h}{l}$ denote this bound.

The lower bound considers that progress has been made only when a pebble arrives at the bottom of hard teeth. Hence,

$$\begin{aligned} & |(\mathcal{F} \cup \mathcal{C}_G) \cap hardteeth_G| \\ & \quad \leq \quad |\mathcal{C}_G \cap hardteeth_G| + |\mathcal{F}| \\ & \quad \leq \quad |\mathcal{C}_G \cap hardteeth_G| + p \end{aligned}$$

is an upper bound on the amount of progress made by the sub-branching program $\widehat{\mathcal{P}}$ on input G. (The lower bound credits the algorithm with p progress, even if the teeth that initially contain pebbles are never traversed to the bottom and even if they are not hard. Because $p << \chi$, this is not a problem.) What remains is to prove that $\widehat{\mathcal{P}}$ makes lots of non-free progress, $|\mathcal{C}_G \cap hardteeth_G|$ is large, for very few comb graph inputs.

Lemma 10 *If $h \leq \frac{|easyteeth_G|}{2}$, then $\Pr_{G \in \mathcal{D}}\left[\, |\mathcal{C}_G \cap hardteeth_G| \geq 2\rho c \,\right] \leq 2^{-0.38\rho c}$, where $\rho = \frac{\chi}{m}$.*

The proof is left until Section 5.8. The idea is as follows. Within the distribution $\mathcal{D}$, the probability of a particular tooth $r \in [1..\chi]$ being hard is $\frac{1}{2}$. However, the NNJAG is not able to move a pebble to a particular tooth. Instead, it must select a connecting edge e_i and move a pebble into what

ever tooth it is attached to. The model can identify the tooth found by the name of its top node. However, the bounded space model cannot have stored very much information about whether or not this tooth is hard. Therefore, the algorithm has little information on which to base the decision as to whether to move the pebble to the bottom of the tooth (i.e. $r \in \mathcal{C}_G$) or not. It turns out that the tooth is hard iff the connecting edge e_i is hard and the probability of this is only $\frac{\chi/2}{m} \approx \rho$, because only $\frac{\chi}{2}$ of the m different connecting edges are chosen to be connected to hard teeth. Using a more formal argument, each of the at most c teeth in $\mathcal{C}_G$ is shown to be hard with probability at most ρ. Hence, we can expect ρc of the teeth in $\mathcal{C}_G$ to be hard. Chernoff's bounds prove that $|\mathcal{C}_G \cap hardteeth_G|$ will not deviate far from this expectation.

The next step is to prove is that if each sub-branching program $\widehat{\mathcal{P}}$ makes sufficient progress for very few inputs, then not too many inputs have a sub-branching program in which sufficient progress is made.

Lemma 11 $\Pr_{G\in\mathcal{D}}\left[\ \exists\widehat{\mathcal{P}}\ \textit{that makes} \geq 2\rho c + p\ \textit{progress for}\ G\right] \leq 2^{(1+o(1))S} \times 2^{-0.38\rho c}$.

Proof of Lemma 11: From Lemma 10, for any sub-branching program $\widehat{\mathcal{P}}$, $\Pr_{G\in\mathcal{D}}\left[\ \widehat{\mathcal{P}}\ \text{that makes} \geq 2\rho c + p \text{ progress for } G\right] \leq 2^{-0.38\rho c}$. The number of nodes in the entire branching program $\mathcal{P}$ and hence the number of sub-branching programs $\widehat{\mathcal{P}}$ is no more than $q \times n^p \times T_{max} \in 2^{(1+o(1))S}$. Thus, the number of inputs that make the stated progress within some sub-branching program $\widehat{\mathcal{P}}$, is no more than $2^{(1+o(1))S}$ times the number that make it with one fixed sub-branching program. The lemma follows. ■

The final step combines Lemma 9 and Lemma 11.

Proof of Theorem 9: Recall, $l = \frac{n}{\chi}$, $h = \frac{\chi}{4}$, $c = \frac{h}{l} = \frac{\chi^2}{4n}$, and $\rho = \frac{\chi}{m}$. Set the number of teeth χ to $2.77 m^{\frac{1}{3}} n^{\frac{1}{3}} S^{\frac{1}{3}}$ in order to insure that $0.38\rho c = 0.38\frac{\chi^3}{4mn} = 2.01\ S$. Finally, set the time bound T_{max} to $0.09\frac{m^{\frac{2}{3}} n^{\frac{2}{3}}}{S^{\frac{1}{3}}}$ or equivalently to $\frac{mn}{4.02\chi} = \frac{ml}{4.02}$ (which is the time for the brute force algorithm). By Lemma 9,

$$\begin{aligned}
&\Pr_{G\in\mathcal{D}}[T_G \leq T_{max} \ \text{ and } \ G \in Corr] \\
&\leq \Pr_{G\in\mathcal{D}}\left[\ \exists\widehat{\mathcal{P}} \text{ that makes } \geq \frac{\frac{\chi}{2}}{\frac{T_{max}}{h}} \text{ progress for } G\right].
\end{aligned}$$

Because $\dfrac{\chi/2}{T_{max}/h} = 2.01\left(\dfrac{\chi}{m}\right)\left(\dfrac{h}{l}\right) = 2.01\rho c \geq 2\rho c + \dfrac{S}{\log n} \geq 2\rho c + p$, it follows that this probability is no more than

$$\begin{aligned}
&\Pr_{G\in\mathcal{D}}\left[\ \exists\widehat{\mathcal{P}} \text{ that makes } \geq 2\rho c + p \text{ progress for } G\right] && \text{(by Lemma 11)} \\
&\leq 2^{(1+o(1))S} \times 2^{-0.38\rho c} && \text{(by the def}^n \text{ of } \chi) \\
&\leq 2^{(1+o(1))S} \times 2^{-2.01S} \leq 2^{-S}. \ \blacksquare
\end{aligned}$$

5.6 A Probabilistic NNJAG with Two Sided Error

We will now consider a probabilistic NNJAG that allows two sided error. A probabilistic algorithm is said to allow **two-sided error** if for every graph it may give the incorrect answer with probability

bounded by $\left(\frac{1}{2}-\epsilon\right)$. The lower bound proves that if the time is limited to only $o\left(\frac{\epsilon^{\frac{4}{3}}m^{\frac{2}{3}}n^{\frac{2}{3}}}{S^{\frac{1}{3}}}\right)$ time steps, then the probabilistic NNJAG cannot do much better than simply always *rejecting* or always *accepting*.

Theorem 10 *There exists a probability distribution $\mathcal{D}'$ on directed graphs so that* $\Pr_{G\in\mathcal{D}'}\left[\, G \in st\text{-}conn \,\right] = \frac{1}{2}$ *and for every probabilistic NNJAG solving directed st-connectivity with two-sided error and* $\epsilon \geq 0$

$$\Pr_{G\in\mathcal{D}'}\left[\, T_G \leq 0.07\frac{\epsilon^{\frac{4}{3}}m^{\frac{2}{3}}n^{\frac{2}{3}}}{S^{\frac{1}{3}}} \;\; and \;\; G \in Corr \,\right] \leq \frac{1}{2}+\epsilon.$$

The proof is similar in structure to that of Theorem 9, but must be changed in two ways. First, the allowed time is restricted to an $\epsilon^{\frac{4}{3}}$ fraction of what it was before so that only an $\frac{\epsilon}{2}$ fraction of the hard teeth are found. Second, the number of easy teeth is restricted to being only an $\frac{\epsilon}{2}$ fraction of the teeth. These conditions ensure that with high probability pebbles reach the bottom of no more than an ϵ fraction of the teeth. This is important because if the NNJAG were to manage to get a pebble to the bottom of more teeth, then it could give the correct answer with probability $> \frac{1}{2}+\epsilon$.

Define the distribution $\mathcal{D}'$ to be the same as $\mathcal{D}$ except for the following changes. First set $|easyteeth_G| = \frac{\epsilon}{2}\chi$ and $|hardteeth_G| = (1-\frac{\epsilon}{2})\,\chi$ and then randomly choose the values for $y_1,..,y_m$ as before. Now decide with probability $\frac{1}{2}$ whether the input will be in st-conn. If it is, then randomly choose **one** of the teeth $r \in [1..\chi]$ and put t on the bottom of this tooth. As before, define T'_{max} to be the smallest integer such that $\Pr_{G\in\mathcal{D}'}\left[\, T_G \leq T'_{max} \;\text{ and }\; G \in Corr \,\right] > \frac{1}{2}+\epsilon$.

Lemma 12

$$\begin{aligned}&\Pr_{G\in\mathcal{D}'}\left[\, T_G \leq T'_{max} \;\; and \;\; G \in Corr \,\right] \\ &\leq \;\; \frac{1}{2}+\frac{\epsilon}{2}+\Pr_{G\in\mathcal{D}'}\left[\, G \text{ makes } \frac{\epsilon}{2}\chi \text{ progress in the first } T'_{max} \text{ time steps} \;\middle|\; G \in \overline{st\text{-}conn} \,\right].\end{aligned}$$

Proof of Lemma 12: For every comb graph $G \notin st$-conn and tooth $r \in [1..\chi]$, define $G|_{\alpha_r=1} \in st$-conn to be the same graph except that t is connected to the bottom of the r^{th} tooth. The domain $\mathcal{D}'$ is partitioned into subdomains to be considered separately. Specifically, define

$\mathcal{A} = \left\{ G \;\middle|\; G \notin st\text{-conn} \text{ and pebbles arrive at the bottom of at least } \frac{\epsilon}{2}\chi \text{ hard teeth during the first } T'_{max} \text{ time steps for } G \right\}$.

$\widehat{\mathcal{A}} = \left\{ G|_{\alpha_r=1} \;\middle|\; G \in \mathcal{A},\ r \in [1..\chi] \right\}$

$\mathcal{B} = \left\{ G \;\middle|\; G \notin st\text{-conn} \text{ and pebbles arrive at the bottom of less than } \frac{\epsilon}{2}\chi \text{ hard teeth during the first } T'_{max} \text{ time steps for } G \right\}$.

$\widehat{\mathcal{B}} = \left\{ G|_{\alpha_r=1} \;\middle|\; G \in \mathcal{B},\ r \in hardteeth_G \text{ whose bottom never contains a pebble during the first } T'_{max} \text{ time steps for } G \right\}$.

$\widehat{\mathcal{B}'} = \left\{ G|_{\alpha_r=1} \;\middle|\; G \in \mathcal{B},\ r \in easyteeth_G \text{ or } r \in hardteeth_G \text{ and a pebble arrives at the bottom of the } r^{th} \text{ tooth during the first } T'_{max} \text{ time steps for } G \right\}$.

The first step is to prove that if $G|_{\alpha_r=1} \in \widehat{\mathcal{B}}$ then $\langle T_G \leq T'_{max} \text{ and } G \in Corr\rangle$ is not true for both it and its unconnected counterpart G. Consider an input $G|_{\alpha_r=1} \in \widehat{\mathcal{B}}$. It follows the same

computation path as the corresponding $G \in \mathcal{B}$ for at least T'_{max} time steps. If the algorithm then continues the computation, then $T_G \leq T'_{max}$ will not be true. Otherwise, by this time the algorithm either has *accepted*, or *rejected*. Either way, the answer is either wrong for $G|_{\alpha_r=1}$ or for G. If the algorithm *rejects* in such situations, then it is correct for $G \in \mathcal{B}$, but not for $G|_{\alpha_r=1} \in \widehat{\mathcal{B}}$. If it *accepts*, then it is correct for $G \in \widehat{\mathcal{B}}$, but not for $G|_{\alpha_r=1} \in \mathcal{B}$. $\Pr_{G\in\mathcal{D}'}\left[\, G \in \mathcal{B} \,\right] \geq \Pr_{G\in\mathcal{D}'}\left[\, G \in \widehat{\mathcal{B}} \,\right]$, because some of the corresponding $G|_{\alpha_r=1}$ inputs are in $\widehat{\mathcal{B}'}$. Hence, it would be wisest for the algorithm to *reject* in such situations. It follows that

$$\begin{aligned}
&\Pr_{G\in\mathcal{D}'}\left[\, T_G \leq T'_{max} \text{ and } G \in Corr \,\right] \\
&\leq \Pr_{G\in\mathcal{D}'}\left[\, G \in \mathcal{A} \cup \widehat{\mathcal{A}} \,\right] + \Pr_{G\in\mathcal{D}'}\left[\, G \in \mathcal{B} \,\right] + \Pr_{G\in\mathcal{D}'}\left[\, G \in \widehat{\mathcal{B}'} \,\right].
\end{aligned}$$

What remains is to bound these probabilities.

Progress is defined to be made when a pebble is moved to the bottom of a hard tooth. Therefore, $\Pr_{G\in\mathcal{D}'}\left[\, G \in \mathcal{A} \,\right] = \Pr_{G\in\mathcal{D}'}\left[\, G \text{ makes } \frac{\epsilon}{2}\chi \text{ progress in the first } T'_{max} \text{ time steps and } G \in \overline{st\text{-conn}} \,\right] = \frac{1}{2} \times \Pr_{G\in\mathcal{D}'}\left[\, G \text{ makes } \frac{\epsilon}{2}\chi \text{ progress in the first } T'_{max} \text{ time steps } \,\middle|\, G \in \overline{st\text{-conn}} \,\right]$. By the definitions, $\Pr_{G\in\mathcal{D}'}\left[\, G \in \widehat{\mathcal{A}} \,\right] = \Pr_{G\in\mathcal{D}'}\left[\, G \in \mathcal{A} \,\right]$. Therefore, $\Pr_{G\in\mathcal{D}'}\left[\, G \in \mathcal{A} \cup \widehat{\mathcal{A}} \,\right] = 2 \times \Pr_{G\in\mathcal{D}'}\left[\, G \in \mathcal{A} \,\right]$.

Bounding $\mathcal{B}$ is easy. It is a subset of $\overline{st\text{-conn}}$ and hence $\Pr_{G\in\mathcal{D}'}\left[\, G \in \mathcal{B} \,\right] \leq \frac{1}{2}$. For each input $G \in \mathcal{B}$, the number of inputs added to $\widehat{\mathcal{B}'}$ is $|easyteeth_G| = \frac{\epsilon}{2}\chi$ plus at most $\frac{\epsilon}{2}\chi$ for the hard teeth. However, the probability of $G|_{\alpha_r=1}$ is $\frac{1}{\chi}$ of the probability of the corresponding G. It follows that $\Pr_{G\in\mathcal{D}'}\left[\, G \in \widehat{\mathcal{B}'} \,\right] \leq \left(\frac{1}{\chi}\right)\left(\frac{\epsilon}{2}\chi + \frac{\epsilon}{2}\chi\right)\Pr_{G\in\mathcal{D}'}\left[\, G \in \mathcal{B} \,\right] \leq \left(\frac{1}{\chi}\right)(\epsilon\chi)\left(\frac{1}{2}\right) \leq \frac{\epsilon}{2}$. The lemma follows. ■

We require a version of Lemma 11 as well.

Lemma 11′ *If* $h \leq \frac{|easyteeth_G|}{2}$, *then* $\Pr_{G\in\mathcal{D}'}\left[\, \exists \widehat{\mathcal{P}} \text{ that makes } \geq 2\rho c + p \text{ progress for } G \,\middle|\, G \in \overline{st\text{-conn}} \,\right] \leq 2^{(1+o(1))S} \times 2^{-0.38\rho c}$.

The fact that $|easyteeth_G|$ and $|hardteeth_G|$ have changed in the definition of $\mathcal{D}'$ changes the parameters but does not change the proof of this lemma. The fact that $\mathcal{D}'$ contains both inputs in st-conn and in $\overline{st\text{-conn}}$ does not change the proof either, because the lemma has been stated in terms of the conditional probability that $G \in \overline{st\text{-conn}}$. Therefore, the proof of this lemma is that same as that in Section 5.8.

Proof of Theorem 10: $l = \frac{n}{\chi}$, $h = \frac{|easyteeth_G|}{2} = \frac{\epsilon}{4}\chi$, $c = \frac{h}{l} = \frac{\epsilon\chi^2}{4n}$, and $\rho = 2\frac{|hardteeth_G|}{m} \leq \frac{2\chi}{m}$. Set $\chi = 1.75\epsilon^{-\frac{1}{3}}m^{\frac{1}{3}}n^{\frac{1}{3}}S^{\frac{1}{3}}$ in order to insure that $0.38\rho c = 0.38\frac{\epsilon\chi^3}{2mn} = 1.01S$. Finally, set the time bound T'_{max} to $0.07\frac{\epsilon^{\frac{4}{3}}m^{\frac{2}{3}}n^{\frac{2}{3}}}{S^{\frac{1}{3}}} = \frac{\epsilon ml}{8.04} = \frac{\epsilon mn}{8.04\chi}$. By Lemma 12,

$$\begin{aligned}
&\Pr_{G\in\mathcal{D}'}\left[\, T_G \leq T'_{max} \text{ and } G \in Corr \,\right] \\
&\leq \frac{1}{2} + \frac{\epsilon}{2} + \Pr_{G\in\mathcal{D}'}\left[\, G \text{ makes } \frac{\epsilon}{2}\chi \text{ progress in the first } T'_{max} \text{ time steps } \,\middle|\, G \in \overline{st\text{-conn}} \,\right]
\end{aligned}$$

$$\leq \frac{1}{2} + \frac{\epsilon}{2} + \Pr_{G \in \mathcal{D}'} \left[\exists \widehat{\mathcal{P}} \text{ that makes } \geq \frac{\frac{\epsilon}{2}\chi}{\frac{T'_{max}}{h}} \text{ progress for } G \;\middle|\; G \in \overline{st\text{-conn}} \right].$$

Because $\frac{\epsilon\chi/2}{T'_{max}/h} = 2.01\left(2\frac{\chi}{m}\right)\left(\frac{h}{l}\right) = 2.01\rho c \geq 2\rho c + \frac{S}{\log n} \geq 2\rho c + p$, it follows that this probability is no more than

$$\begin{aligned}
&\frac{1}{2} + \frac{\epsilon}{2} + \Pr_{G \in \mathcal{D}'} \left[\exists \widehat{\mathcal{P}} \text{ that makes } \geq 2\rho c + p \text{ progress for } G \;\middle|\; G \in \overline{st\text{-conn}} \right] \qquad \text{(by Lemma 11}') \\
&\leq \frac{1}{2} + \frac{\epsilon}{2} + 2^{(1+o(1))S} \times 2^{-0.38\rho c} \qquad \text{(by the def}^n \text{ of } \chi) \\
&\leq \frac{1}{2} + \frac{\epsilon}{2} + 2^{(1+o(1))S} \times 2^{-1.01S} \\
&\leq \frac{1}{2} + \epsilon. \quad \blacksquare
\end{aligned}$$

What remains is to prove Lemma 10. This requires the use of Chernoff bounds.

5.7 Trials

A well known theorem by Chernoff [Sp92] is that given a number of independent trials, the probability of having twice the expected number of successes is exponentially small.

Lemma 13 *Suppose that $x_r \in \{0,1\}$ is 1 with probability ρ independently for each $r \in C$. Then for every $\epsilon > 0$, $\Pr\left[\left|\sum_{r\in C} x_r - \rho|C|\right| \geq \epsilon\rho|C|\right] \leq 2^{-k_\epsilon \rho|C|}$,*
where $k_\epsilon = \min\left(-\ln\left(e^\epsilon (1+\epsilon)^{-(1+\epsilon)}\right), \frac{\epsilon^2}{2}\right)$.

For example, for $\epsilon = 1$, $k_\epsilon > 0.38$. Moreover, for every $\delta \geq 1$, $\Pr\left[\sum_{r\in C} x_r \geq 2\delta\rho|C|\right] \leq 2^{-0.38\delta\rho|C|}$. This becomes more complicated if the trials are not independent. However, if each trial has low probability of success no matter what outcomes of the other trails have, then the lemma still holds.

Lemma 14 *For each $r \in C$, let $\widehat{x}_r \in \{0,1\}$ be the random variable indicating the success of the r^{th} trial. For each $r \in C$ and $\mathcal{O} \in \{0,1\}^{C-\{r\}}$, let $Z_{\langle r,\mathcal{O}\rangle} = \Pr\left[\widehat{x}_r = 1 \;\middle|\; \mathcal{O}\right]$, where $\mathcal{O}$ indicates that the other trials have the stated outcome. If $\forall r, \mathcal{O}$, $Z_{\langle r,\mathcal{O}\rangle} \leq \rho$, then for every $\delta \geq 1$, $\Pr\left[\sum_{r\in C} \widehat{x}_r \geq 2\delta\rho|C|\right] \leq 2^{-0.38\delta\rho|C|}$*

The following proof by Hisao Tamaki [Tam93] uses the fact the $\widehat{x}_r$ events are probabilistically dominated by the x_r events.

Proof of Lemma 14 [Tam93]: It is sufficient to prove that for all trials $r \in [1..|C|]$ and for all integers k,

$$\Pr\left[\sum_{i\in[1..r]} \widehat{x}_i \geq k\right] \leq \Pr\left[\sum_{i\in[1..r]} x_i \geq k\right].$$

This is proved by induction on r. It is trivially true for $r = 0$. Now assume it is true for $r-1$. Then

$$\begin{aligned}
&\Pr\left[\sum_{i\in[1..r]} \widehat{x}_i \geq k\right] \\
&= \Pr\left[\sum_{i\in[1..r-1]} \widehat{x}_i \geq k\right] + \Pr\left[\widehat{x}_r = 1 \;\middle|\; \sum_{i\in[1..r-1]} \widehat{x}_i = k-1\right] \times \Pr\left[\sum_{i\in[1..r-1]} \widehat{x}_i = k-1\right] \\
&\leq \Pr\left[\sum_{i\in[1..r-1]} x_i \geq k\right] + \rho \times \Pr\left[\sum_{i\in[1..r-1]} x_i = k-1\right] \\
&= Pr\left[\sum_{i\in[1..r]} x_i \geq k\right]. \blacksquare
\end{aligned}$$

5.8 The Probability of Finding a Hard Tooth

The goal of this section is to prove that lots of non-free progress is made by the sub-branching program $\widehat{\mathcal{P}}$, on very few comb graph inputs. Namely,

Lemma 10 *If* $h \leq \frac{|easyteeth_G|}{2}$, *then* $\Pr_{G\in\mathcal{D}}\left[\,|\mathcal{C}_G \cap hardteeth_G| \geq 2\rho c\,\right] \leq 2^{-0.38\rho c}$, *where* $\rho = \frac{\chi}{m}$.

Proof of Lemma 10: Note that the set of teeth $\mathcal{C}_G$ depends on the input G only as far as which computation path γ it follows. Therefore, it is well defined to instead refer to the set $\mathcal{C}_\gamma$. Because every input follows one and only one computation path γ through $\widehat{\mathcal{P}}$, it is sufficient to prove that for every path γ, a lot of progress is made for very few of the inputs that follow the computation path γ. Specifically, for each path γ through $\widehat{\mathcal{P}}$, we prove that $\Pr_{G\in\mathcal{D}}\left[\,|\mathcal{C}_\gamma \cap hardteeth_G| \geq 2\rho c \;\middle|\; G \text{ follows } \gamma\,\right] \leq 2^{-0.38\rho c}$.

Each tooth $r \in \mathcal{C}_\gamma$ can be thought of as a trial. The r^{th} trial consists of choosing a random input G subject to the condition that G follows the computation path γ. The trial succeeds if the r^{th} tooth is hard. For each trial $r \in \mathcal{C}_\gamma$ and each $\mathcal{O} \in \{succeeds, fails\}^{\mathcal{C}_\gamma - \{r\}}$, let $Z_{\langle\gamma,r,\mathcal{O}\rangle} = \Pr_{G\in\mathcal{D}}\left[\, r \in hardteeth_G \;\middle|\; G \text{ follows } \gamma \text{ and } \text{ satisfies } \mathcal{O}\,\right]$, where $\mathcal{O}$ indicates the condition that the other trials have the stated outcome. We will proceed to prove that for each r and each $\mathcal{O}$, $Z_{\langle\gamma,r,\mathcal{O}\rangle} \leq \frac{\chi}{m} = \rho$.

In order to bound $Z_{\langle\gamma,r,\mathcal{O}\rangle}$, fix $r \in \mathcal{C}_\gamma$ and the outcomes $\mathcal{O} \in \{0,1\}^{\mathcal{C}_\gamma - \{r\}}$ for the other trials. The first step is to understand the condition "G follows γ". Recall that each node of a branching program queries at most one of the variables $y_1, \ldots, y_m, \alpha_1, \ldots, \alpha_\chi$ and the input G follows the branch corresponding to the value learned. Therefore, a computation path γ can be specified by stating the collection of variables queried and their values. For example, $\gamma = \{y_j = r, \ldots, y_{j'} = r', \alpha_{r''} = 0, \ldots, \alpha_{r'''} = 0\}$. Recall, that in the input distribution $\mathcal{D}$, only inputs $G \notin st\text{-conn}$ are considered, i.e. for all $r \in [1..\chi]$, $\alpha_r = 0$. Therefore, in all the paths γ considered, this will be the case. Hence, the α's will be omitted.

Because $r \in \mathcal{C}_\gamma$, we know that at the beginning of the sub-branching program $\widehat{\mathcal{P}}$, the r^{th} tooth does not contain a pebble and at some point in the computation a pebble arrives at the bottom of this tooth. Therefore, by Lemma 8, we know a pebble must enter the r^{th} tooth via one of the

connecting edges during the computation path γ. Without loss of generality, let the connecting edge in question be e_j. Recall that when a pebble walks the connecting edge e_j into the top of the r^{th} tooth, the branching program learns that $y_j = r$. Hence, the condition that "G follows γ" includes the condition that $y_j = r$.

How does this condition by itself affect the probability that r is in $hardteeth_G$? From the definition of $hardedges_G$, we know that if $y_j = r$, then $y_j \in hardedges_G$ if and only if $r \in hardteeth_G$. This gives us that $\Pr_{G \in \mathcal{D}}\left[r \in hardteeth_G \;\middle|\; y_j = r \right] = \Pr_{G \in \mathcal{D}}\left[y_j \in hardedges_G \;\middle|\; y_j = r \right]$. This is equal to $\Pr_{G \in \mathcal{D}}\left[y_j \in hardedges_G \right]$, because we know that y_j has some value and there is a symmetry amongst all the possible values. Hence, telling you that $y_j = r$ gives you no information about whether $y_j \in hardedges_G$. Finally, $\Pr_{G \in \mathcal{D}}\left[y_j \in hardedges_G \right] = \frac{\chi/2}{m}$, because $\frac{\chi}{2}$ of the variables $y_1, \ldots, y_m$ are randomly chosen to be in $hardedges_G$.

What remains is to consider the additional effects of the other conditions. Note that if γ fixes more than one y variable to r, then $\Pr_{G \in D}\left[r \in hardteeth_G \;\middle|\; G \text{ follows } \gamma \right] = 0$, which is trivially $\leq \rho$. Therefore, suppose that γ does not assign any other variable to r and let $\widehat{\gamma} = \{y_{j'} = r', \ldots, y_{j''} = r''\}$ be all the assignments of the y variables made by γ, excluding y_j. Hence,

$$\begin{aligned} Z_{\langle \gamma, r, \mathcal{O} \rangle} &= \Pr_{G \in D}\left[r \in hardteeth_G \;\middle|\; y_j = r \text{ and } \widehat{\gamma} \text{ and } \mathcal{O} \right] \\ &= \frac{\Pr_{G \in D}\left[r \in hardteeth_G \text{ and } y_j = r \text{ and } \widehat{\gamma} \text{ and } \mathcal{O} \right]}{\Pr_{G \in D}\left[y_j = r \text{ and } \widehat{\gamma} \text{ and } \mathcal{O} \right]} \end{aligned}$$

Define $\mathcal{R}_{\widehat{\gamma}} \subseteq [1..\chi] - \{r\}$ to be the teeth that are included in the computation path $\widehat{\gamma} = \{y_{j'} = r', \ldots, y_{j''} = r''\}$. There is nothing special about the tooth r except that it is not mentioned in $\widehat{\gamma}$ or in $\mathcal{O}$. Therefore, for all $r' \notin \mathcal{R}_{\widehat{\gamma}}$,

$$\begin{aligned} &\Pr_{G \in D}\left[r \in hardteeth_G \text{ and } y_j = r \text{ and } \widehat{\gamma} \text{ and } \mathcal{O} \right] \\ &= \Pr_{G \in D}\left[r' \in hardteeth_G \text{ and } y_j = r' \text{ and } \widehat{\gamma} \text{ and } \mathcal{O} \right] \text{ and} \\ &\Pr_{G \in D}\left[y_j = r \text{ and } \widehat{\gamma} \text{ and } \mathcal{O} \right] = \Pr_{G \in D}\left[y_j = r' \text{ and } \widehat{\gamma} \text{ and } \mathcal{O} \right]. \end{aligned}$$

From this we get that

$$\begin{aligned} Z_{\langle \gamma, r, \mathcal{O} \rangle} &= \frac{\frac{1}{\chi - |\mathcal{R}_{\widehat{\gamma}}|} \sum_{r' \notin \mathcal{R}_{\widehat{\gamma}}} \Pr_{G \in D}\left[r' \in hardteeth_G \text{ and } y_j = r' \text{ and } \widehat{\gamma} \text{ and } \mathcal{O} \right]}{\frac{1}{\chi - |\mathcal{R}_{\widehat{\gamma}}|} \sum_{r' \notin \mathcal{R}_{\widehat{\gamma}}} \Pr_{G \in D}\left[y_j = r' \text{ and } \widehat{\gamma} \text{ and } \mathcal{O} \right]} \\ &= \frac{\Pr_{G \in D}\left[y_j \in hardedges_G \text{ and } y_j \text{ is not assigned a tooth in } \mathcal{R}_{\widehat{\gamma}} \text{ and } \widehat{\gamma} \text{ and } \mathcal{O} \right]}{\Pr_{G \in D}\left[y_j \text{ is not assigned a tooth in } \mathcal{R}_{\widehat{\gamma}} \text{ and } \widehat{\gamma} \text{ and } \mathcal{O} \right]} \\ &= \frac{\Pr_{G \in D}\left[y_j \in hardedges_G \text{ and } y_j \text{ is not assigned a tooth in } \mathcal{R}_{\widehat{\gamma}} \;\middle|\; \widehat{\gamma} \text{ and } \mathcal{O} \right]}{\Pr_{G \in D}\left[y_j \text{ is not assigned a tooth in } \mathcal{R}_{\widehat{\gamma}} \;\middle|\; \widehat{\gamma} \text{ and } \mathcal{O} \right]} \end{aligned} \tag{5.1}$$

The following claims bound certain probabilities.

Claim 4 $\Pr_{G\in D}\left[\, y_j \in \mathit{hardedges}_G \;\middle|\; \widehat{\gamma} \text{ and } \mathcal{O} \,\right] \leq \dfrac{\chi}{2\,(m-h)}$.

Proof of Claim 4: The condition $\mathcal{O}$ fixes for each tooth in $\mathcal{C}_\gamma - \{r\}$ whether it is easy or hard. Define $\mathcal{O}'$ so that it fixes this for all of the teeth $[1..\chi]$ in a way that is consistent with $\mathcal{O}$. Define $\mathcal{Y}_{\widehat{\gamma}} \subseteq \{y_1, \ldots, y_m\}$ to be the variables that are assigned values in the computation path $\widehat{\gamma} = \{y_{j'} = r', \ldots, y_{j''} = r''\}$ and partition these variables into $\mathcal{Y}_{\widehat{\gamma},e}$ and $\mathcal{Y}_{\widehat{\gamma},h}$ according to which are fixed to teeth that $\mathcal{O}'$ fixes to be easy and which that it fixes to be hard. Note y_j is contained in neither of these sets.

Conditioning on the fact that the variables in $\mathcal{Y}_{\widehat{\gamma},e}$ are easy and those in $\mathcal{Y}_{\widehat{\gamma},h}$ are hard effects the probability that y_j is hard. However, beyond this, the conditions $\widehat{\gamma}$ and $\mathcal{O}'$ do not effect this probability. Which of the variables $y_1, \ldots, y_m$ are chosen to be in $\mathit{hardedges}_G$ is independent which teeth are easy or hard and hence independent of $\mathcal{O}'$. Once a variable in $\mathcal{Y}_{\widehat{\gamma},e}$ is fixed to be easy, it is assigned a tooth independent of whether y_j is hard. Similarly, for the variables in $\mathcal{Y}_{\widehat{\gamma},h}$. Hence, the fact that $\widehat{\gamma}$ fixes the teeth assigned to the variables in $\mathcal{Y}_{\widehat{\gamma},e} \cup \mathcal{Y}_{\widehat{\gamma},h}$ does not effect this probability, beyond fixing whether the variables are hard or easy. It follows that $\Pr_{G\in D}\left[\, y_j \in \mathit{hardedges}_G \;\middle|\; \widehat{\gamma} \text{ and } \mathcal{O}' \,\right] = \Pr_{G\in D}\left[\, y_j \in \mathit{hardedges}_G \;\middle|\; \mathcal{Y}_{\widehat{\gamma},e} \text{ are easy and } \mathcal{Y}_{\widehat{\gamma},h} \text{ are hard} \,\right]$.

Recall that according to the distribution $\mathcal{D}$, $\frac{\chi}{2}$ of the variables $y_1, \ldots, y_m$ are randomly chosen to be in $\mathit{hardedges}_G$. If we know that those in $\mathcal{Y}_{\widehat{\gamma},e}$ are easy and $\mathcal{Y}_{\widehat{\gamma},h}$ are hard, then what remains is to choose $\frac{\chi}{2} - |\mathcal{Y}_{\widehat{\gamma},h}|$ of the $m - |\mathcal{Y}_{\widehat{\gamma},e}| - |\mathcal{Y}_{\widehat{\gamma},h}|$ variables to be the remaining hard variables. The variable $y_j \notin \mathcal{Y}_{\widehat{\gamma}}$ is one of these to be selected from. It follows that $\Pr_{G\in D}\left[\, y_j \in \mathit{hardedges}_G \;\middle|\; \mathcal{Y}_{\widehat{\gamma},e} \text{ are easy and } \mathcal{Y}_{\widehat{\gamma},h} \text{ are hard} \,\right] = \frac{\chi/2 - |\mathcal{Y}_{\widehat{\gamma},h}|}{m - |\mathcal{Y}_{\widehat{\gamma},e}| - |\mathcal{Y}_{\widehat{\gamma},h}|}$. Note that $|\mathcal{Y}_{\widehat{\gamma},e}| + |\mathcal{Y}_{\widehat{\gamma},h}| = |\mathcal{Y}_{\widehat{\gamma}}| \leq h$, because $\widehat{\mathcal{P}}$ is of height h. The worst case is when $|\mathcal{Y}_{\widehat{\gamma},e}| = h$ and $|\mathcal{Y}_{\widehat{\gamma},h}| = 0$. giving that $\Pr_{G\in D}\left[\, y_j \in \mathit{hardedges}_G \;\middle|\; \widehat{\gamma} \text{ and } \mathcal{O}' \,\right] \leq \frac{\chi}{2(m-h)}$. Because this is true for every extension $\mathcal{O}'$, it is also true for $\mathcal{O}$. ■

Claim 5 $\Pr_{G\in D}\left[\, y_j \textit{ is assigned a tooth in } \mathcal{R}_{\widehat{\gamma}} \;\middle|\; y_j \notin \mathit{hardedges}_G \textit{ and } \widehat{\gamma} \textit{ and } \mathcal{O} \,\right] \leq \dfrac{1}{2}$.

Proof of Claim 5: Suppose that the teeth are partitioned into $\mathit{hardteeth}_G$ and $\mathit{easyteeth}_G$ according to $\mathcal{O}$, the connecting edges are partitioned into $\mathit{hardedges}_G$ and $\mathit{easyedges}_G$ fixing y_j to be in $\mathit{easyedges}_G$, and the variables $y_1, \ldots, y_m$ other than y_j have been assigned teeth according to $\widehat{\gamma}$. The variable y_j is then given a random value from $\mathit{easyteeth}_G$. The probability that it is assigned a tooth from the set $\mathcal{R}_{\widehat{\gamma}}$ is $\frac{|\mathcal{R}_{\widehat{\gamma}} \cap \mathit{easyteeth}_G|}{|\mathit{easyteeth}_G|} \leq \frac{|\mathcal{R}_{\widehat{\gamma}}|}{|\mathit{easyteeth}_G|} \leq \frac{1}{2}$, because $|\mathcal{R}_{\widehat{\gamma}}| \leq h = \frac{|\mathit{easyteeth}_G|}{2}$. ■

These two claims can be combined to give

$$
\begin{aligned}
&\Pr_{G\in D}\left[\, y_j \notin \mathit{hardedges}_G \text{ and } y_j \text{ is not assigned a tooth in } \mathcal{R}_{\widehat{\gamma}} \;\middle|\; \widehat{\gamma} \text{ and } \mathcal{O} \,\right] \\
&\quad = \Pr_{G\in D}\left[\, y_j \text{ is not assigned a tooth in } \mathcal{R}_{\widehat{\gamma}} \;\middle|\; y_j \notin \mathit{hardedges}_G \text{ and } \widehat{\gamma} \text{ and } \mathcal{O} \,\right] \\
&\qquad \times \Pr_{G\in D}\left[\, y_j \notin \mathit{hardedges}_G \;\middle|\; \widehat{\gamma} \text{ and } \mathcal{O} \,\right] \\
&\quad \geq (1 - 1/2) \times \left(1 - \frac{\chi}{2\,(m-h)}\right) \qquad (5.2)
\end{aligned}
$$

To conclude, note that the denominator of (5.1) is sum of the numerator and (5.2). Also note that the numerator $\Pr_{G\in D}\left[\, y_j \in hardedges_G \text{ and } y_j \text{ is not assigned a tooth in } \mathcal{R}_{\widehat{\gamma}} \;\middle|\; \widehat{\gamma} \text{ and } \mathcal{O}\,\right] \leq \Pr_{G\in D}\left[\, y_j \in hardedges_G \;\middle|\; \widehat{\gamma} \text{ and } \mathcal{O}\,\right] \leq \frac{\chi}{2(m-h)}$. This gives

$$Z_{\langle\gamma,r,\mathcal{O}\rangle} \leq \frac{\frac{\chi}{2(m-h)}}{\frac{\chi}{2(m-h)} + \left(1 - \frac{\chi}{2(m-h)}\right)/2} = \frac{\chi}{m-h+\frac{\chi}{2}} \leq \frac{\chi}{m} = \rho.$$

We are now able to complete the proof of the lemma. Each tooth $r \in \mathcal{C}_\gamma$ is thought of as a trial. The r^{th} trial consists of choosing a random input G subject to the condition that G follows the computation path γ. The trial succeeds if the r^{th} tooth is hard. For each trial $r \in \mathcal{C}_\gamma$ and each $\mathcal{O} \in \{0,1\}^{\mathcal{C}_\gamma - \{r\}}$, $Z_{\langle\gamma,r,\mathcal{O}\rangle} = \Pr_{G\in\mathcal{D}}\left[\, r \in hardteeth_G \;\middle|\; G \text{ follows } \gamma \text{ and satisfies } \mathcal{O}\,\right] \leq \frac{\chi}{m} = \rho$. The total number of successes is $|\mathcal{C}_\gamma \cap hardteeth_G|$. Therefore, by Lemma 14, $\Pr_{G\in\mathcal{D}}\left[\, |\mathcal{C}_\gamma \cap hardteeth_G| \geq 2\delta\rho|\mathcal{C}_\gamma| \;\middle|\; G \text{ follows } \gamma\,\right] \leq 2^{-0.38\delta\rho|\mathcal{C}_\gamma|}$. (The fact that the probabilities in the statement of the lemma are conditioned on $\langle G \text{ follows } \gamma\rangle$ does not effect the result, because every probability is conditioned in the same way.) Let $\delta = \frac{c}{|\mathcal{C}_\gamma|}$. Section 5.5 proved that $|\mathcal{C}_\gamma| \leq \frac{h}{l} = c$. Therefore, $\delta \geq 1$ and hence $\Pr_{G\in\mathcal{D}}\left[\, |\mathcal{C}_\gamma \cap hardteeth_G| \geq 2\rho c \;\middle|\; G \text{ follows } \gamma\,\right] \leq 2^{-0.38\rho c}$. ■

Chapter 6

Future Work

In this chapter, I outline the directions in which the work in this thesis might be improved, describe some possible approaches and discuss some of their limitations.

6.1 Undirected Graphs

For undirected graphs, the future goals are to increase the number of pebbles beyond $O\left(\frac{\log n}{\log\log n}\right)$ and to increase the amount of time beyond $n \times 2^{\Omega\left(\frac{\log n}{\log\log n}\right)}$. Two approaches are to use the fly swatter techniques and to use the comb graph techniques.

The lower bound for the recursive fly swatter graph of $n \times 2^{\Omega\left(\frac{\log n}{\log\log n}\right)}$ is tight for a helper-JAG even with worst case analysis, when all $s_{\langle l,i\rangle}, t_{\langle l,i\rangle}$ nodes are distinguished. This could be done by removing the helper or by making the $s_{\langle l,i\rangle}, t_{\langle l,i\rangle}$ nodes indistinguishable. It is my belief that the complexity on a regular JAG is actually $\Theta\left(n^2\right)$. The helper-JAG algorithms arise from the surprising protocol for the helper-parity game, in which the helper compacts a tremendous amount of useful information about the input into very few bits. I conjecture that a JAG is not able to do this compacting in $o\left(n^2\right)$ time.

Some recent work suggests that the result can be improved possibly even $n^{2-\epsilon}$ by making the nodes $s_{\langle l,i\rangle}$ and $t_{\langle l,i\rangle}$ indistinguishable. With this change, a JAG is unable to "ask a parity question" with one pebble, because when the pebble comes down the handle it does not know whether $s_{\langle l,i\rangle}$ or $t_{\langle l,i\rangle}$ is reached. Moreover, the JAG does not know which direction he is going in the line of d fly swatters and hence is effectively taking a random walk on the line. In general, a random walk of length d takes d^2 time, however, the JAG does have some information about the direction traveled. Hence, it is not totally clear what result will be obtained. Possibly it can be proved that the time to traverse the line is bounded by $d^{2-\epsilon}$. This would give a result of $\Omega\left(n^{2-\epsilon}\right)$.

It is unlikely that a lower bound with the number of pebbles increased beyond $\log n$ can be proved using the fly swatter graph. The technique, as does the Cook and Rackoff proof [CR80], allows for an extra pebble for each level of recursion. A graph with n nodes cannot be constructed with more than $\log_2 n$ levels of recursion. The only non-trivial lower bound for more than $O\left(\log n\right)$ jumping pebbles is the lower bound for comb graphs given in Chapters 4 and 5. These are however for directed st-connectivity. There are a number of problems with trying to extend these techniques

to undirected graphs. The first issue is that on an undirected version of the comb graph, pebbles would be able to trivially travel from t backwards to s. This problem can be solved by have in two identical copies of the graph connected at the t nodes. The new questions is whether s and s' are connected. The advantage of this is that all pebbles then are placed on one of the s nodes. Another problem is that the JAG can differentiate between the teeth by learning the degrees of their top nodes. In the JAG (but not the NNJAG) setting, I am able to give the JAG a comb graph for which these degrees are all the same. This is done by proving a lower bound on the partition game when the input is guaranteed to be a partition of the m connecting edges into χ parts of equal sides. (This is non-trivial.) A third problem arrises from the labels on the edges from the tops of the teeth to the back nodes. In fact, the following is a 3 pebble, linear time deterministic algorithm for undirected comb graphs.

For each back node v_i
  move two pebbles P_1 and P_2 to v_i
  move P_1 into tooth connected to v_i
  move P_1 back up to the back nodes through the edge labeled 1.
  If P_1 finds P_2 then
    we know that the edged connected to v_i has label 1
    move P_1 back into the tooth,
    down to the bottom,
    look for node t
  end if
end for

Every tooth is connected to some back node v_i via the edge labeled 1. Therefore, every tooth will be traversed once and only once.

6.2 Directed Graphs

Recently Greg Barnes and I [BE93] increased the time-space tradeoff for directed graphs from $T \times S^{\frac{1}{3}} \in \Omega\left(n^{\frac{4}{3}}\right)$ to $T \times S \in \frac{n^2}{\log^{O(1)} n}$ on a NNJAG. The techniques are similar to those in Chapter 4, but use a more complicated family of graphs.

Although there are algorithms for undirected st-connectivity with $T \times S \in \Omega\left(n^2 \log^{O(1)} n\right)$, for the directed case, known algorithms that use sub-linear space, use much more than n^2 time. Therefore, it might be possible to increase the time-space tradeoff to $\omega\left(n^2\right)$ and show that directed st-connectivity is strictly harder than undirected st-connectivity. However, neither the proof by Cook and Rackoff [CR80] nor those in Chapters 4 and 5 are able to do this since for the families of directed graphs considered, st-connectivity can be solved in $O\left(n^2\right)$ time. The techniques used in Chapters 4 and 5 only use the fact that, for directed graphs, the model is not able to follow edges backwards. Cook and Rackoff's JAG space lower bound [CR80] use this fact as well, but in a very different way. They use the fact that when a pebble traverses a directed edge, it cannot get back unless there is a pebble left above it for it to jump to. The techniques used for the comb graph, on the other hand, could let the pebbles retrace their steps. They use the fact that the JAG with a pebble on a node $u_{\langle r,1\rangle}$ cannot know what or how many edges point to this node. Therefore, the JAG does not know whether it has already done the work of traversing down the r^{th} tooth.

The last and most significant future goal is to prove the same lower bounds on a general model of computation. Recall that the restriction on the NNJAG is that it lacks random access to its input. I am able to push the techniques used in Chapter 5 further, by extending the $T \times S^{\frac{1}{3}} \in \Omega\left(n^{\frac{4}{3}}\right)$ lower bound to a model that allows random access to its input. This model is an (r-way) branching program with a reasonable restriction on what information the model can store. I can also prove this bound for the unrestricted (r-way) branching program, but for a computation problem similar to st-connectivity that requires the computation to output for each tooth whether s is connected to t via this tooth. The main difficulty in proving a lower bound on a branching program for a decision problem seems to be defining a good measure of the progress of a computation.

Chapter 7

Glossary

JAG and NNJAG model (Section 1.1)

G	input graph
n	number of nodes in the graph
m	number of edges in the graph
s	distinguished node of input graph to st-connectivity
t	distinguished node of input graph to st-connectivity
v	arbitrary node of graph
p	number of pebbles
$P \in [1..p]$	a pebble
q	number of states
$Q \in [1..q]$	a state
$\Pi \in [1..n]^p$	specifies the location of each pebble
$R \in \{0,1\}^*$	random bit string used
T	computation time or a specific time step
$S = p \log_2 n + \log_2 q$	space used by the JAG
one-sided error	the probabilistic algorithm must give the correct answer for every input in the language
two-sided error	the probabilistic algorithm must give the correct answer with high probability for every input
log	all logarithms are base 2

The Helper-Parity Game (Chapter 2)

r	number of bits of input per game instance
d	number of game instances
b	number of bits of help from the helper
$\vec{\alpha} = \langle \alpha_1, \ldots, \alpha_d \rangle \in (\{0,1\}^r - \{0^r\})^d$	input to helper-parity game

$M(\vec{\alpha})$	message sent by helper in helper-parity game
$c_{\langle\vec{\alpha},i\rangle}$	the number of questions asked about the i^{th} game instance on input $\vec{\alpha}$
$c_{\vec{\alpha}}$	$\frac{1}{d}\sum_{i\in[1..d]} c_{\langle\vec{\alpha},i\rangle}$

Recursive line of fly swatter (Chapter 3)

H	the traversal graph of a st-connectivity computation
r	number of switches per fly swatter graph
$\frac{h}{2}$	length of the handle
d	number of fly swatters per line
L	number of levels of recursion
l	a specific level of recursion
$\vec{\alpha}_l = \left\langle \alpha_{\langle l,1\rangle}, \ldots, \alpha_{\langle l,d\rangle} \right\rangle \in (\{0,1\}^r - \{0^r\})^d$	indicates switches for the l^{th} level
$\vec{\gamma} = \langle \vec{\alpha}_1, \ldots, \vec{\alpha}_{l-1} \rangle$	the structure of the "super edges"
$\vec{\beta} = \langle \vec{\alpha}_{l+1}, \ldots, \vec{\alpha}_L \rangle$	the "context" in which the line of fly swatter appears
$G\left(\vec{\gamma}, \vec{\alpha}_l, \vec{\beta}\right)$	the recursive fly swatter graph

Comb Graph (Chapter 4-5)

$v_1, \ldots, v_n$	the back nodes of the comb graph
$e_1, \ldots, e_m$	the connecting edges
χ	the number of teeth
$l = \frac{n}{\chi}$	the length of the teeth
r	indexes a specific tooth
$u_{\langle r,1\rangle}, \ldots, u_{\langle r,l\rangle}$	the r^{th} tooth
$y_1, ..y_m \in [1..\chi]$	specifies which tooth the connecting edge e_i is connected to
$\alpha_r \in \{0,1\}$	specifies whether the bottom of tooth r is connected to t

NNJAG Lower Bound (Chapter 5)

$\mathcal{D} \subseteq \overline{st\text{-conn}}$	probability distribution on graphs
$easyteeth_G$	set of teeth attached to many connecting edges
$hardteeth_G$	set of teeth attached to few connecting edges
$easyedges_G$	set of connecting edges attached to teeth in $easyteeth_G$
$hardedges_G$	set of connecting edges attached to teeth in $hardteeth_G$
$\rho = \frac{\chi}{n}$	probability $y_i \in hardedges_G$
Progress	the number of hard teeth that have had a pebble on its bottom node at some point in time so far

$T_{\langle G,R\rangle}$	the computation time for input G and random bits R
$Corr$	set of $\langle G, R\rangle$ for which the correct answer is given
$\mathcal{P}$	branching program for random string $R \in \{0,1\}^*$
$\langle Q, \Pi, T\rangle$	configuration node of $\mathcal{P}$
$\widehat{\mathcal{P}}$	sub-branching program of $\mathcal{P}$
$h = \frac{\lvert easyteeth_G\rvert}{2} = \frac{\chi}{4}$	height of sub-branching program
$\mathcal{F} \subseteq [1..\chi]$	the set of teeth that contain pebbles at the root of $\widehat{\mathcal{P}}$, $\lvert\mathcal{F}\rvert \leq p$
$\mathcal{C}_G \subseteq [1..\chi]$	the set of teeth that do not contain pebbles at the root of $\widehat{\mathcal{P}}$, yet whose bottoms contain a pebble at some point during the computation by $\widehat{\mathcal{P}}$ on G
$c = \frac{h}{l}$	upper bound on $\lvert\mathcal{C}_G\rvert$
computation path	path traced by input through a branching program
$\gamma = \{y_j = r, \ldots, y_{j'} = r', \alpha_{r''} = 0, \ldots, \alpha_{r'''} = 0\}$	computation path through $\widehat{\mathcal{P}}$
$\widehat{\gamma} = \{y'_j = r', \ldots, y_{j''} = r''\}$	be all the assignments of the y variable in γ, excluding those that are assigned r
$\mathcal{R}_{\widehat{\gamma}} \subseteq [1..\chi] - \{r\}$	the teeth that are included in $\widehat{\gamma}$
$\mathcal{O} \in \{0,1\}^{\mathcal{C}_\gamma - \{r\}}$	outcome of the trials other then r

Bibliography

[Ab86] K. Abrahamson. Time-space tradeoffs for branching programs contrasted with those for straight-line programs. In *27th Annual Symposium on Foundations of Computer Science*, pages 402–409, Toronto, Ontario, October 1986.

[Ad78] L. Adleman. Two theorems on random polynomial time. In *19th Annual Symposium on Foundations of Computer Science*, pages 75–83, Ann Arbor, MI, October 1978. IEEE.

[AKLLR79] 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.

[BNS89] László Babai, Noam Nisan, and Márió Szegedy. Multiparty protocols, pseudorandom generators for logspace, and time-space trade-offs. *Journal of Computer and System Sciences*, 45(2):204–232, October 1992.

[BBRS92] G. Barnes, J. Buss, W. Ruzzo, and B .Schieber. A sublinear space, polynomial time algorithm for directed *s-t* connectivity. In *Proceedings of the Seventh Annual Conference, Structure in Complexity Theory*, pages 27–33, Boston, MA, June 1992.

[BE93] Greg Barnes, Jeff Edmonds. Time-Space Lower Bounds for Directed *s-t* Connectivity on JAG Models To appear in *34st Annual Symposium on Foundations of Computer Science*, Palo Alto, CA, Nov. 1993.

[BF93] G. Barnes, and U. Feige. Short random walks on graphs. In *Proceedings of the Twenty Fifth Annual ACM Symposium on Theory of Computing*, San Diego, CA, May 1993.

[Be91] P. Beame. A general time-space tradeoff for finding unique elements. *SIAM Journal on Computing*, 20(2):270–277, 1991.

[BBRRT90] 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. Full version submitted for journal publication.

[BS83] 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.

[Bo82] Allan Borodin. Structured vs. general models in computational complexity. *L'Enseignement Mathématique*, XXVIII(3-4):171–190, July-December 1982.

[BC82] A. Borodin and S. A. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM Journal on Computing*, 11(2):287–297, May 1982.

[BFMUW87] A. Borodin, F. Fich, F. Meyer auf der Heide, E. Upfal, and A. Wigderson. A time-space tradeoff for element distinctness. *SIAM Journal on Computing*, 16(1):97–99, February 1987.

[BFKLT81] A. Borodin, M. J. Fischer, D. G. Kirkpatrick, N. A. Lynch, and M. Tompa. A time-space tradeoff for sorting on non-oblivious machines. *Journal of Computer and System Sciences*, 22(3):351–364, June 1981.

[BRT92] 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.

[BKRU89] 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.

[CRRST89] 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.

[Co66] Alan Cobham. The recognition problem for the set of perfect squares. Research Paper RC-1704, IBM Watson Research Center, 1966.

[CR80] 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.

[Ed93a] 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*, page 718-727, San Diego, CA, May 1993.

[Ed93b] Jeff Edmonds. Trading non-deterministic help for deterministic time in multiple instances of a game. Manuscript.

[Ed93c] Jeff Edmonds. $\Omega\left(n^{4/3}\right)$ time-space trade-offs for directed *st*-connectivity on a JAG and other more powerful models. Submitted to *34st Annual Symposium on Foundations of Computer Science*, Palo Alto, CA, Nov. 1993.

[Im93] Russell Impagliazzo. Personal conversation.

[IZ89] 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.

[KLNS89] 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.

[Ni92] Noam Nisan. *RL* $\subseteq$ *SC*. In *Proceedings of the Twenty Fourth Annual ACM Symposium on Theory of Computing*, pages 619–623, Victoria, B.C., Canada, May 1992.

[NSW92] 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.

[Po93a] C. K. Poon. A sublinear space, polynomial time algorithm for directed *st*-connectivity on the JAG model. Manuscript.

[Po93b] C. K. Poon. Space Bounds For Graph Connectivity Problems On Node-named JAGs and Node-ordered JAGs. To appear in *34st Annual Symposium on Foundations of Computer Science*, Palo Alto, CA, Nov. 1993.

[R93] Steven Rudich. personal communication.

[Sa70] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[Sa73] W. J. Savitch. Maze recognizing automata and nondeterministic tape complexity. *Journal of Computer and System Sciences*, 7(4):389–403, 1973.

[Sp92] N. A. Spencer. The probabilistic Method. *John Wiley and Sons, Inc.*, page 239, 1992.

[Tam93] Hisao Tamaki. personal communication.

[Tar72] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, June 1972.

[Tm80] Martin Tompa. Time-space tradeoffs for computing functions, using connectivity properties of their circuits. *Journal of Computer and System Sciences*, 20:118–132, April 1980.

[Wi92] A. Wigderson. The Complexity of Graph Connectivity. *Proceedings of the 17th Symposium on the Mathematical Foundations of Computer Science*, 1992.

[Ye84] Y. Yesha. Time-space tradeoffs for matrix multiplication and the discrete Fourier transform on any general sequential random-access computer. *Journal of Computer and System Sciences*, 29:183–197, 1984.

[Ya77] 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.

[Ya88] A. C. Yao. Near-optimal time-space tradeoff for element distinctness. In *29th Annual Symposium on Foundations of Computer Science*, pages 91–97, White Plains, NY, October 1988. IEEE.