Tight Lower Bounds for *st*-Connectivity on the NNJAG Model

Jeff Edmonds [*] (jeff@cs.yorku.ca)
Chung Keung Poon [†] (ckpoon@cs.cityu.edu.hk)
Dimitris Achlioptas [‡] (optas@cs.toronto.edu)

Abstract

Directed *st*-connectivity is the problem of deciding whether or not there exists a path from a distinguished node s to a distinguished node t in a directed graph. We prove a time-space lower bound on the probabilistic NNJAG model of Poon [Poo93]. Let n be the number of nodes in the input graph and S, T be the space and time used by the NNJAG, respectively. We show that for any $\delta > 0$ if an NNJAG uses space $S \in O(n^{1-\delta})$ then $T \in 2^{\Omega(\log^2(n/S))}$, otherwise $T \in 2^{\Omega(\log^2(\frac{n \log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$. (In a preliminary version of this paper by Edmonds and Poon [EP95], a lower bound of $T \in 2^{\Omega(\log^2(\frac{n \log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$ was proved.) Our result greatly improves the previous lower bound of $ST \in \Omega(n^2/\log n)$ on the JAG model by Barnes and Edmonds [BE93] and that of $S^{1/3}T \in \Omega(n^{4/3})$ on the NNJAG model by Edmonds [Edm93a]. Our lower bound is tight for $S \in O(n^{1-\delta})$, for any $\delta > 0$, matching the upper bound of Barnes et al. [BBRS92]. As a corollary of this improved lower bound we obtain the first tight space lower bound of $\Omega(\log^2 n)$ on the NNJAG model. No tight space lower bound was previously known even for the more restricted JAG model.

1 Introduction

The *st*-connectivity problem (STCON) is a fundamental problem in computer science as it is the natural abstraction of many search processes. Its space and time-space complexities are of special interest because there are many applications such as game searching, program verification and databases in which the size of the input graph is too large compared to the size of the internal memory of a machine. In these applications algorithms that run in small space, and preferably in small time simultaneously, are required. STCON is also important in computational complexity theory

[*]Department of Computer Science, York University, Toronto, ON M3J 1P3, Canada. Major portion of this work was done while the author was at the International Computer Science Institute, Berkeley, USA.

[†]Department of Computer Science, City University of Hong Kong. This work is partially supported by Texas Advanced Research Projects Grant 003658386. Major portion of it was done while the author was at Dept. of Computer Science, University of Toronto, Canada.

[‡]Department of Computer Science, University of Toronto, Toronto, ON M5S 3G4, Canada.

as it is complete for NSPACE($\log n$) under logspace reductions. Both STCON and the corresponding problem for undirected graphs, USTCON, are hard for DSPACE($\log n$) as any problem solvable deterministically in logarithmic space can be reduced to either problem. (See Lewis and Papadimitriou [LP82] and Savitch [Sav70].) Thus, showing that there is no deterministic logarithmic space algorithm for STCON would separate the classes DSPACE($\log n$) and NSPACE($\log n$), while devising such an algorithm would prove that DSPACE($f(n)$) = NSPACE($f(n)$) for any space-constructible function $f(n) \in \Omega(\log n)$ [Sav70]. STCON is also a candidate problem for separating the classes of SC and NC [Joh90]. Below we mention the previous works that are most relevant to our paper. For more information on graph connectivity, we refer the reader to the beautiful survey paper by Wigderson [Wig92].

1.1 Previous works

The most common algorithms for *st*-connectivity, breadth- and depth-first search run in optimal time $O(m+n)$ and use $O(n \log n)$ space. At the other extreme, Savitch [Sav70] provided an algorithm that uses $O(\log^2 n)$ space and requires time exponential in its space bound (i.e., time $n^{O(\log n)}$). Tompa [Tom82] showed that STCON cannot be solved in polynomial time and sub-linear space simultaneously by the repeated squaring method. However, Barnes et al. [BBRS92] gave a polynomial time algorithm for STCON that uses space $S \in n/2^{\Theta(\sqrt{\log n})}$, providing the first polynomial time, sub-linear space algorithm. This shows that the repeated squaring method is too restricted. In fact, their algorithm implies a general time-space upper bound of $T \in 2^{O(\log^2(\frac{n \log n}{S}))} \times n^3$ for $S \in \Omega(\log^2 n)$.

A natural question is whether the upper bounds of Savitch and Barnes et al. are tight. Unfortunately, proving non-trivial lower bounds for natural decision problems on any general model of computation, such as Turing machines and branching programs appears to be beyond the reach of current techniques. Thus, it is natural to consider *structured* computational models [Bor82] whose basic operations are based on the structure of the input, as opposed to being based on the bits in the input's encoding. A natural structured model for STCON is the "jumping automaton for graphs", or *JAG*, introduced by Cook and Rackoff [CR80]. A JAG moves a set of pebbles on the graph. There are two basic operations — moving a pebble along a directed edge in the graph, and jumping a pebble from its current location to the node occupied by another pebble. Although the JAG model is structured, it is powerful enough to simulate most known algorithms for STCON and related problems. For example, depth-first and breadth-first search, random walks [AKL+79] and the algorithms of Savitch and Barnes et al. can all be simulated on a JAG, see [CR80] and [Poo96]. To the authors' knowledge, all known deterministic or probabilistic algorithms for directed STCON are implementable on a JAG. However, it is not clear how a nondeterministic JAG can simulate Immerman's and Szelepcsényi's $O(\log n)$-space algorithm for directed *st*-nonconnectivity

($\overline{\text{STCON}}$) [Imm88, Sze88]. This motivated Poon [Poo93] to introduce the more general Node-Named JAG (*NNJAG*) model, an extension of the JAG where the computation is allowed to depend on the names of the nodes on which the pebbles are located. Using this added power, Poon [Poo93] showed how to simulate the Immerman / Szelepcsényi algorithm on a nondeterministic NNJAG.

Cook and Rackoff [CR80] proved a lower bound of $\Omega(\log^2 n/\log\log n)$ on the space required for a JAG to compute STCON. Within the $\log\log n$ factor, this is tight with Savitch's algorithm. Berman and Simon [BS83] extended this result to the probabilistic JAG model. More precisely, they showed that any probabilistic JAG that solves STCON within $2^{\log^{O(1)} n}$ expected time requires $\Omega(\log^2 n/\log\log n)$ space. Their probabilistic JAG is allowed to flip a coin in each step and is able to solve STCON with 1-sided error using $O(\log n)$ space and $O(n^n)$ expected time, see Gill [Gil77]. In the following, we will refer to such a probabilistic machine as a coin-flipping machine. Poon [Poo93] further generalized the bound showing that $S \in \Omega(\frac{\log^2 n}{\log\log n+\log\log T})$ for any coin-flipping probabilistic NNJAG with space S and expected time T.

Regarding the time-space tradeoff, there are many lower bounds proved for USTCON on various weaker variants of the JAG model [BBR+90, BRT92, CR80]. Edmonds [Edm93b] was the first to prove a time-space lower bound for USTCON on the regular JAG model (with bounded space). All these results apply to (directed) STCON, which contains USTCON as a special case. However, USTCON appears to be easier than STCON both in terms of space and time-space complexity. For example, Nisan et al. [NSW92] showed that USTCON can be solved in $O(\log^{1.5} n)$ space on a deterministic Turing machine. There is also a randomized $O(\log n)$ space, polynomial time algorithm (by Aleliunas et al. [AKL+79]) and a deterministic $O(\log^2 n)$ space, polynomial time algorithm (by Nisan [Nis92]) for this problem. Although it is not known whether the algorithms in [NSW92, Nis92] can be simulated on a JAG or NNJAG, USTCON can indeed be solved in $O(\log n)$ space and polynomial time on a JAG due to the existence of polynomial length universal traversal sequences [AKL+79]. Thus one cannot hope to get super-polynomial time lower bounds for STCON by establishing similar bounds for USTCON.

The first nontrivial lower bound explicitly for STCON was given by Barnes and Edmonds [BE93]. They showed that $ST \in \Omega(n^2/\log n)$ on the JAG model. In fact their result was proved on a more powerful variant of JAG called *many states, big step JAG* which, unlike an ordinary JAG, is capable of traversing trees in $O(\log n)$ space. Using a proof technique completely different from [BE93], Edmonds [Edm93a] showed that $S^{1/3}T \in \Omega(n^{4/3})$ on the NNJAG model. These results still do not yield super-polynomial lower bounds on time no matter how small S is. In view of this large gap between the upper and lower bounds and the fact that the Barnes et al. algorithm was obtained by combining several rather simple ideas, it seemed that further improvements to the upper bound were quite possible.

1.2 New results

Rather surprisingly, in a preliminary version of this paper by Edmonds and Poon [EP95], a lower bound of $T \in 2^{\Omega(\log^2(\frac{n \log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$ is obtained. This implies that super-polynomial running time is necessary to solve the problem whenever S is smaller than $(n \log n)/2^{\omega(\sqrt{\log n \cdot \log\log n})}$. The bound also nearly matches the upper bound of $T \in 2^{O(\log^2(\frac{n \log n}{S}))} \times n^3$ (which is super-polynomial for $S \in (n \log n)/2^{\omega(\sqrt{\log n})}$) by Barnes et al. [BBRS92]. Here, by a more careful choice of parameters and a tighter analysis, we prove that for any $\delta > 0$, a probabilistic NNJAG with 2-sided error, using space $S \in O(n^{1-\delta})$, requires expected time $T \in 2^{\Omega(\log^2(n/S))}$, matching the upper bound of [BBRS92].

In this paper, we define an S-space probabilistic NNJAG as a distribution of S-space deterministic NNJAGs. Hence the probabilistic NNJAG must use time $T \in 2^{O(S)}$ or else it will cycle. From this fact and the time-space tradeoff, we obtain the first tight space lower bound of $\Omega(\log^2 n)$ on a probabilistic NNJAG with 2-sided error. No tight space lower bound was previously known even for the more restricted JAG model. On the other hand, a coin-flipping probabilistic JAG or NNJAG (as defined in [BS83, Poo93]) can run usefully for up to $2^{2^{O(S)}}$ expected time. As mentioned before, it can solve STCON with $O(\log n)$ space and $O(n^n)$ expected time. Thus, one can only prove a time-space lower bound on this coin-flipping model. Since a coin-flipping probabilistic NNJAG with space S and time T can be simulated on our probabilistic NNJAG using time T and space $S + \log T$, our result is valid on the coin-flipping model for $S \in \Omega(\log^2 n)$ (since $\log T \in O(S)$). For space $S \in O(\log^2 n)$, our result still implies a lower bound of $T \in 2^{\Omega(\log^2 n)}$ on the coin-flipping model. However, for $S \in O(\frac{\log^2 n}{\log\log n})$, Poon [Poo93] gives a stronger lower bound of $T \in 2^{(2^{\Omega(\log^2 n/S)})}$. For example, when $S \in O(\log n)$, his result implies that $T \in 2^{n^c}$, for some constant $c > 0$.

This paper borrows a lot of techniques from [Edm93a]. The bound is proved for the probabilistic NNJAG model by transforming the machine into a structured branching program, and applying a progress argument introduced by Borodin et al. [BFK+81] and also used in many proofs of time-space tradeoff lower bounds, including [BC82, Bea91, BFMadH+87, Yao88]. Roughly, the argument is that for every short path of the computation, the probability that lots of progress is made, conditional on the fact that this computation path is followed, is less than 2^{-S} (with space S there are at most 2^S different such sub-computations). Our proof, however, is complicated by the fact that this is not true for some "lucky" computation paths and hence a number of new techniques are required to overcome this. In addition, the argument is applied recursively yielding a substantially greater lower bound than what would be possible without recursion. We note that similar recursive techniques have also been used in [CR80, BS83, Yao88, Edm93b, Poo93].

1.3 Organization of this paper

We first define the NNJAG model in Section 2. In Section 3, we give the statement of our main result and its corollaries. In Sections 4 and 5, we describe the families of graphs used to defeat the NNJAG. In Section 6, we define a notion of progress for an NNJAG on such families of graphs. In Section 7, we enhance and stylise the NNJAG model to simplify our proof. Sections 8 to 12 contain the technical proof of the lower bound. Section 8 contains the proof of an inductive statement, Lemma 8.3, from which our main result follows. The proof makes forward references to Lemma 8.1 and Lemma 8.2 which are proved in Sections 10 to 12, and in Section 9 respectively. Section 13 gives the conclusion and some open problems.

2 The NNJAG model

A (deterministic) NNJAG [Poo93] J is a finite state automaton with p distinguishable pebbles, q states and a transition function Δ. The transition function Δ can depend non-uniformly on the size, n, of the input graph and the values of p, q can be functions of n. The input to J is a triple (G, s, t) where G is an n-node graph containing nodes s and t. For every node in G, its out-edges are labelled with consecutive integers starting at 0. The nodes in G are also labelled from 0 up to $n-1$. We define the *instantaneous description* (id) of J as the pair (Q, Π) where Q is the current state and Π is a mapping of pebbles to nodes, specifying the current location of each pebble in the graph. When J is in id (Q, Π), the transition function Δ determines the next move for J based on (1) the state Q and (2) the mapping Π. A move is either a *walk* or a *jump*. A walk (P, i, Q') consists of moving pebble P along the edge labelled i that comes out of the node $\Pi(P)$ and then assuming state Q'. (If there is no such edge, the pebble just remains on the same node.) A jump (P, P', Q') consists of moving pebble P to the node $\Pi(P')$ and then assuming state Q'. The NNJAG J is initialized to state Q_0 with all its pebbles on node s. It is said to *accept* an input (G, s, t) if it enters an accepting state on this input. An NNJAG solves STCON for n-node graphs if for every input (G, s, t) where G is an n-node directed graph, it accepts the input if and only if there is a directed path from s to t in G. We define the space used by the NNJAG as $p \log n + \log q$, i.e. as the number of bits needed to specify an id. The time used is the number of moves it has made. For simplicity, we assume that the labels of nodes s and t are always fixed (say, as 0 and $n-1$ respectively). Hence s and t are not part of the input.

A probabilistic NNJAG J is defined as a distribution on deterministic NNJAGs. On a given input, it first chooses probabilistically a deterministic NNJAG from the distribution and then runs this deterministic NNJAG on the input. The space used is taken as the maximum over all the deterministic NNJAGs in the distribution and the expected (worst case) time is the expected (worst case) running time over the distribution. We say that J solves STCON with 2-sided error if

for every input (G, s, t), the probability of J entering an accepting state is at least $3/4$ when there is a path from node s to t, and at most $1/4$ otherwise.

3 Statement of Results

Our main result is the following.

Theorem 3.1 *If J is a probabilistic NNJAG that solves* STCON *on n-node graphs while taking expected time T and using space S, then $T \in 2^{\Omega(\log^2(n/S))}$ when $S \in O(n^{1-\delta})$, where $\delta > 0$, and $T \in 2^{\Omega(\log^2(\frac{n \log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$ otherwise.*

The proof of Theorem 3.1 follows by applying "Yao's lemma" [Yao77] to the following theorem.

Theorem 3.2 *For any δ, $\epsilon > 0$ there is a distribution $\mathcal{D}$ on n-node graphs such that*

1. *$Pr_{G \in \mathcal{D}}[G \in$ STCON $] = 1/2$, and*
2. *for any deterministic NNJAG using space $S \in O(n^{1-\delta})$ and (worst case) time $T \notin 2^{\Omega(\log^2(n/S))}$, or $S \in \omega(n^{1-\delta})$ and $T \notin 2^{\Omega(\log^2(\frac{n \log n}{S})/\log\log n))} \times (nS/\log n)^{1/2}$*

$$Pr_{G \in \mathcal{D}}[J \text{ is correct on input } G] < \frac{1}{2} + 2\epsilon.$$

Proof of Theorem 3.1: Theorem 3 of [Yao77] states that for any randomized algorithm $\mathcal{J}$ that has probability of error at most λ and any input distribution $\mathcal{D}$, the expected time of $\mathcal{J}$ on the worst case input is at least half of the average time of the best deterministic algorithm that errs with probability at most 2λ on random input chosen from $\mathcal{D}$. By Theorem 3.2, the latter quantity is at least $T \times (1 - 2\lambda - \frac{1}{2} - 2\epsilon)$ where $T \in 2^{\Omega(\log^2(n/S))}$ for $S \in O(n^{1-\delta})$ and $T \in 2^{\Omega(\log^2(\frac{n \log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$ otherwise. Putting λ as some constant less than $\frac{1}{4} - \epsilon$ and since $S \in O(n^{1-\delta})$, for some $\delta > 0$, we get the required lower bound on a probabilistic NNJAG that errs with probability at most λ. ∎

Theorem 3.2 is strong enough to yield an optimal space lower bound for the deterministic NNJAG model, as an immediate corollary.

Corollary 1 *Any probabilistic NNJAG that solves* STCON *requires $\Omega(\log^2 n)$ space.*

Proof: Once the deterministic NNJAG to be used is chosen from the distribution the probabilistic NNJAG becomes deterministic. Hence, while using space at most S, the NNJAG cannot take more than $2^{O(S)}$ steps without going into an infinite loop. If an NNJAG J uses space $S \notin \Omega(\log^2 n)$ then, for sufficiently large n, the number of steps it can take is smaller than the lower bound implied by Theorem 3.2 and the result follows. ∎

4 Layered Graphs

From now on, we let δ be a fixed positive constant. A (d, x, f)-*layered graph*, first defined in [BE93], is a graph consisting of d layers, each containing x nodes. The j^{th} node in layer i is denoted by (and named) $u_{\langle i,j\rangle}$. (Hence, the Node Named JAG always knows the location of a pebble in terms of i, j.) Every node has at most f out-going edges to some (not necessarily distinct) nodes in the next layer. Here, we will set $f = \Theta((n \log n/S)^{1/2})$ for $S \in O(n^{1-\delta})$ and $f = 2$ otherwise.

Let $D = \lceil 80 \log n / \log f \rceil$ (so that $f^D \geq n^{80}$). Note that D is constant with respect to n if $S \in O(n^{1-\delta})$ and $D \in \Theta(\log n)$ otherwise. The distribution $\mathcal{B}(x)$ is a distribution on (D, x, f)-layered graphs. Each graph $G \in \mathcal{B}(x)$ will have $x/2$ *hard paths* (to be defined shortly) of length D and is obtained as follows. In each layer i, except the top layer, we pick (without replacement) a sequence of $x/2$ nodes, uniformly at random. Let us denote the j^{th} node picked as $v_{\langle i,j\rangle}$. (It is the node $u_{\langle i,j'\rangle}$ for some j'.) These nodes are called the *hard nodes*. The remaining $x/2$ nodes in that layer are called the *easy nodes*. For layer 1, we choose the sequence of nodes $u_{\langle 1,1\rangle}, u_{\langle 1,2\rangle}, \ldots, u_{\langle 1,x/2\rangle}$ as the sequence of hard nodes. We shall put in edges so that if an NNJAG walks a pebble $D-1$ steps starting from a hard node in the top layer, it is difficult for the pebble to be on a hard node when it reaches layer D.

First, the hard nodes are connected by the edges $(v_{\langle i,j\rangle}, v_{\langle i+1,j\rangle})$ for each $i \in [1..D-1]$ and each $j \in [1..x/2]$. The path from $v_{\langle 1,j\rangle}$ to $v_{\langle D,j\rangle}$ is called the j^{th} *hard path*. The nodes $v_{\langle 1,j\rangle}$ and $v_{\langle D,j\rangle}$ are called the *root* and *goal* of the j^{th} hard path respectively. Thus, there are $x/2$ hard paths, roots, and goals in G. The edge labels are chosen independently and uniformly from $[0..f-1]$. Thus, for each root r the vector of edge labels on the hard path rooted at r, denoted by $\vec{\ell}_r$, is chosen uniformly at random from $[0..f-1]^{D-1}$.

For each layer $i \in [1..D-1]$, each hard node $v_{\langle i,j\rangle}$ will have another $f-1$ out-going edges and each easy node will have f out-going edges. The destinations of these edges are chosen independently (with replacement) at random from the set of easy nodes in layer $i+1$. In this way, the in-degree of each hard node is kept to one.

5 Recursively Layered Graphs

Set $\chi = \Theta((\frac{n^3 S}{\log n})^{1/4})$ for $S \in O(n^{1-\delta})$ and $\chi = \Theta((\frac{nS}{\log n})^{1/2})$ otherwise. Set $K = \lfloor \frac{\log(n/(4\chi))}{\log 2D} \rfloor$. Thus $K \in \Theta(\log(\frac{n \log n}{S}))$ for $S \in O(n^{1-\delta})$ and $K \in \Theta(\log(\frac{n \log n}{S})/\log\log n)$ otherwise. Moreover, $K \leq \log n$ since $S \geq \log n$. We first construct, recursively, $K+1$ distributions $\mathcal{H}_0, \mathcal{H}_1, \ldots, \mathcal{H}_K$ on layered graphs where $\mathcal{H}_k$ is a distribution on $(D^k, 2^k\chi, f)$-layered graphs. Each such graph has χ *super goals*. In addition, for $k > 0$, each graph in $\mathcal{H}_k$ has $D^{k-1}2^{k-1}\chi$ hard paths of length D each one with a goal. Our input distribution $\mathcal{D}$ of n-node graphs in Theorem 3.2 is formed by adding a few nodes and edges to each graph in $\mathcal{H}_K$.

The distribution $\mathcal{H}_0$ contains only one graph which is simply a layer of χ isolated nodes. These nodes are the super goals. For $k > 0$, a graph G in $\mathcal{H}_k$ is formed as follows. We choose a graph G' from $\mathcal{H}_{k-1}$ and replace each layer i of G' with a graph G_i chosen from $\mathcal{B}(2^k\chi)$. Note that each G_i has $2^k\chi/2 = 2^{k-1}\chi$ hard paths and each layer of G' has the same number of nodes. We identify the j^{th} hard path of G_i (i.e., the path from $v_{\langle 1,j\rangle}$ to $v_{\langle D,j\rangle}$ of G_i) with the j^{th} node in layer i (i.e., $u_{\langle i,j\rangle}$) of G'. Every edge that goes into $u_{\langle i,j\rangle}$ of G' will now go into $v_{\langle 1,j\rangle}$ of G_i and every edge that goes out of $u_{\langle i,j\rangle}$ of G' will go out of $v_{\langle D,j\rangle}$ of G_i. The easy nodes in G_i are not connected to any node outside G_i.

Since G is uniquely determined by G' and $G_1, \ldots, G_{D^{k-1}}$ (and vice-versa), we often denote G as a tuple $\langle G_1, G_2, \ldots, G_{D^{k-1}}; G'\rangle$. The graph G' is called the *collapsed* graph of G, denoted by $C(G)$. The set of hard paths (respectively, roots and goals) of G is the union of all the sets of hard paths (respectively, roots and goals) in $G_1, G_2, \ldots, G_{D^{k-1}}$. Hence G has $D^{k-1}2^{k-1}\chi$ hard paths (and the same number of roots and goals) in total. The χ super goal nodes in G are the goal nodes of the χ hard paths in $G_{D^{k-1}}$, representing the χ super goal nodes in $G' \in \mathcal{H}_{k-1}$. Note that the super goals are on the bottom level of G and they are associated with the χ nodes in the graph from $\mathcal{H}_0$. The edges within each of the G_is are called the *base edges* of G. The other edges, i.e., those connecting the G_is, are in 1-1 correspondence with the edges in the collapsed graph $C(G)$ of G and hence they are called the *collapsed edges*. Note that graphs in $\mathcal{H}_1$ have base edges but not collapsed edges, and the graph in $\mathcal{H}_0$ does not have any edge at all.

Figure 1 shows a graph $G \in \mathcal{H}_k$ on the right and on the left its collapsed graph $C(G) \in \mathcal{H}_{k-1}$ and the symbol for a base graph in $\mathcal{B}(2^k\chi)$. We rearranged the nodes so that all the hard nodes in the G_is appear on the left half.

For each $k \in [0..K]$, we obtain a distribution $\mathcal{G}_k$ by adding to each graph in $\mathcal{H}_k$ the following *auxiliary* nodes and edges (see Figure 2):

(A1) a directed path $(s = w_1, w_2, \ldots, w_{2^k\chi})$ with $w_1 = s$ and for each $j \in [1..2^k\chi]$, an edge from w_j to $u_{\langle 1,j\rangle}$ of G,

(A2) the isolated node t,

(A3) a special isolated node, referred to as the *lost* node.

(A4) a number of isolated nodes so that the total number of nodes in the graph is exactly n.

The lost node is introduced for technical reasons that will become clear in Section 7. These auxiliary nodes and edges are fixed for each graph $G \in \mathcal{G}_k$. Hence for $k > 0$, G can still be specified by a tuple $\langle G_1, G_2, \ldots, G_{D^{k-1}}; G'\rangle$ where $G_1, \ldots, G_{D^{k-1}}$ are in $\mathcal{B}(2^k\chi)$ and G' is in $\mathcal{H}_k$. The *collapsed* graph of G, denoted by $C(G)$, is the graph G' augmented with the auxiliary nodes and edges needed to form a graph in $\mathcal{G}_{k-1}$ from a graph in $\mathcal{H}_{k-1}$. Thus $C(G)$ is in $\mathcal{G}_{k-1}$. Note that excluding the nodes added in (A4), each graph in $\mathcal{G}_k$ consists of a $(D^k, 2^k\chi, f)$-layered graph,

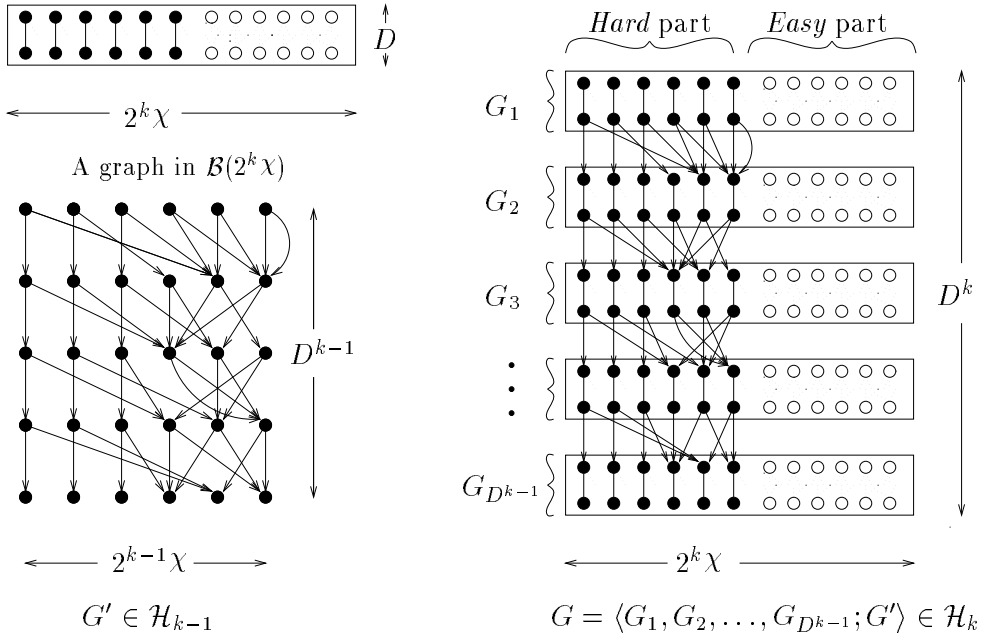

Figure 1: An example where $f = 2$.

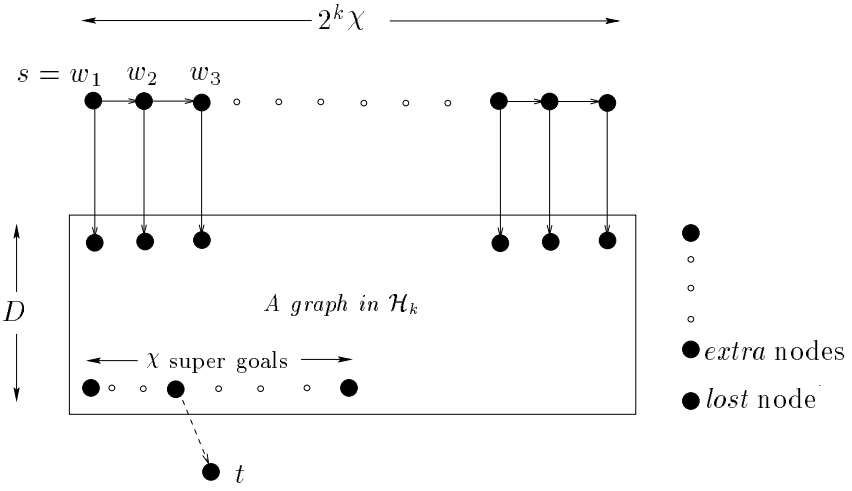

Figure 2: A graph in $\mathcal{G}_k$

a path with $2^k\chi$ nodes, the node t and the lost node. These add up to a total of $(2D)^k\chi + 2^k\chi + 2 \leq 4(2D)^k\chi \leq n$ nodes for $k \leq K$ by our choice of χ and K. Hence, we are not adding a negative number of nodes in (A4). Finally, the distribution $\mathcal{D}$ of Theorem 3.2 is defined as follows. First choose a graph $G' \in \mathcal{G}_K$ and then uniformly at random choose one of the χ super goals in G' as the *special node*. With probability 1/2 connect the special node to the isolated node t to form a graph G. Clearly, $\Pr_{G \in \mathcal{D}}[G \in \text{STCON}] = 1/2$.

6 Defining Progress

Consider the computation of an NNJAG J on input G. We will analyse the progress of J during different phases of the computation. In the following definition, a *subcomputation* A refers to a sequence of moves taken by the NNJAG starting from certain id (Q, Π). Once we recast an NNJAG as a branching program in Section 7, one can think of A as a sub-branching program.

Definition 1 *For any subcomputation A and any input $G \in \mathcal{G}_k$, $w_A(G)$ is the number of different goals in G that were pebbled (i.e., reached by a pebble) at any time during A. Similarly, $w^*_A(G)$ is the number of super goals in G that were pebbled during A.*

Note that when A begins, some pebbles may already be sitting on a goal node. These goals will be counted as progress in $w_A(G)$. However, there can be at most $S/\log n$ such progress. The following lemma shows why reaching the other goals is difficult for an NNJAG.

Lemma 6.1 *If at some step T', a particular hard path does not contain any pebble, and at some later step T'', a pebble arrives at the goal of this path then each edge in that path must be traversed by some pebble between step T' and T''.*

Proof: Observe that every node on a hard path has in-degree one and in the NNJAG model a pebble can arrive at a node only if the node is already occupied by some pebble or if it walks to the node. ∎

We point out that it is not neccessary for a general computation model to find out the hard path before it can inspect the edge connections of the associated goal node. This is the only significant difference between a general model and an NNJAG that we will employ in our proof.

Recall that an input $G = \langle G_1, \ldots, G_{D^{k-1}}; G' \rangle \in \mathcal{G}_k$ consists of the collapsed graph $G' \in \mathcal{G}_{k-1}$ and the base graphs $G_1, \ldots, G_{D^{k-1}} \in \mathcal{B}(2^k\chi)$. The NNJAG has to learn both the structure of the base graphs and that of the collapsed graph. Obviously, $w_A(G)$ measures how much A has learnt about the base graphs. The following lemma shows that $w_A(G)$ is also a good estimate of the number of different collapsed edges traversed during a subcomputation A.

Lemma 6.2 *The number of different collapsed edges of an input graph $G \in \mathcal{G}_k$ that can be traversed during a subcomputation A of an NNJAG is at most $f \times w_A(G)$.*

Proof: An NNJAG can traverse an edge (u, v) only if there is a pebble on node u before the traversal. If the edge is a collapsed edge, u must be a goal. Since every goal has out-degree at most f, pebbling one such node allows the NNJAG to traverse at most f different collapsed edges. ∎

Lemma 8.3, to follow, uses lemmata 6.1 and 6.2 recursively to prove that it is hard for an NNJAG to reach the χ super goal nodes. Roughly speaking, the argument goes as follows: Suppose we have proved that it is hard to visit the super goals of graphs chosen from $\mathcal{G}_{k-1}$ within time T_{k-1}. Consider a graph $G = \langle G_1, \ldots, G_{D^{k-1}}; G' \rangle$ in $\mathcal{G}_k$ and an NNJAG J with time T_k. We will prove, using Lemma 6.1, that for any input $G' \in \mathcal{G}_{k-1}$, it is hard for J to visit many goals in the graphs $G_1, \ldots, G_{D^{k-1}} \in \mathcal{B}(2^k\chi)$ within time T_k. In particular, Lemma 6.2 implies that no more than T_{k-1} different edges in G' are traversed. To conclude the argument, we show that J is effectively an NNJAG trying to reach, within time T_{k-1}, the super goals for graphs chosen from $\mathcal{G}_{k-1}$, which is difficult by the inductive assumption.

It should be pointed out that J can traverse the same edge many times (which is natural as J cannot remember the result of too many edge traversals with limited space). Therefore, we cannot directly claim that J runs in T_{k-1} time on inputs from $\mathcal{G}_{k-1}$. For this reason, we measure the time of an NNJAG using the *s-height*, $h_A()$, of the corresponding branching program A. Precise definitions of s-height will be given in Section 7. Here, we just state that an NNJAG running in time T will have $h_A(G) \leq T$ for any G. Thus Lemma 8.3 will imply that if J is an NNJAG that uses space $S \in O(n^{1-\delta})$ and time $T \notin 2^{\Omega(\log^2(n/S))}$ or space $S \in \omega(n^{1-\delta})$ and time $T \notin 2^{\Omega(\log^2(\frac{n \log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$, then for any $\epsilon > 0$, $\Pr_{G \in \mathcal{D}}[\, w^*_J(G) > \epsilon\chi\,] < \epsilon$. Below we show how Theorem 3.2 follows from this last statement.

Proof of Theorem 3.2: Choose $G \in \mathcal{D}$. Recall that this can be done by choosing $G_a \in \mathcal{G}_K$ and then choosing one of its χ supergoals to be special, uniformly at random. Let G_b be the same as G_a except with an edge from the special node to t. Then G is uniformly chosen to be G_a or G_b. If G_a is such that $w^*_J(G_a) > \epsilon\chi$, i.e., J reaches a lot of super goals, then assume that J gives the correct answer on G. From Lemma 8.3, the probability of this event is less than ϵ. If J pebbles at most $\epsilon\chi$ super goals, then the probability that J pebbles the special node is at most ϵ because the NNJAG cannot tell that a super goal is special unless it pebbles it. Finally, if J does not pebble the special node it cannot learn whether there is an edge from the special node to t. Therefore, in this case, the computations on G_a and G_b are the same and hence the probability of giving the correct answer for G is $1/2$. Thus, the probability of giving the right answer for G is less than $1/2 + 2\epsilon$. ∎

7 An NNJAG as a Branching Program

We will introduce a variant of the NNJAG model which we call the *pebble location redundant* NNJAG model. The reason for this is that while the new model maintains all the power of an NNJAG it

helps us prove a collapsing lemma. In particular, we shall see that it is helpful to construct a pebble location redundant NNJAG J' for graphs in $\mathcal{G}_{k-1}$ from a pebble location redundant NNJAG J for graphs in $\mathcal{G}_k$. We call this the "collapsing" of J to J'.

An NNJAG is said to be pebble location redundant if the current state always determines the current location of all the pebbles and, hence, the state alone is sufficient to specify the id of the NNJAG. More formally, this means that there is a function $\widehat{\Pi}$ such that if the NNJAG is in state Q, then $\widehat{\Pi}(Q)$ specifies the locations of all the pebbles. As a first step in getting a pebble location redundant NNJAG we enhance a standard NNJAG as follows. First, we allow it to jump a pebble to the lost node (which is isolated), and for any $j \in [1..2^k\chi]$ to the nodes w_j and $u_{\langle 1,j\rangle}$. We call such a jump a *node-jump*. Note that in the standard NNJAG model a pebble can only jump to (the node occupied already by) another pebble. Also, we modify a step to be taken from an id (Q,Π) by the NNJAG to consist of the following substeps:

(Substep 1) Based on (Q,Π), it either walks a pebble P along the edge with a specified label ℓ, or it node-jumps a pebble P. It can also choose not to move any pebble. Let Π_1 specify the new pebble locations.

(Substep 2) Based on (Q,Π) and Π_1, it performs a (possibly empty) sequence of pebble-to-pebble jumps and then assumes some state Q'.

The intuition supporting these modifications is that a sequence of moves of a standard NNJAG can be viewed as a sequence of "macro steps", each of which starts with a walk, followed by a (possibly empty) sequence of jumps. Each such jump causes the standard NNJAG to enter a unique next id. Intuitively, the NNJAG "learns" about the input only by taking walking steps. Each macro step can be performed in one step in the enhanced model. It follows that a time lower bound on this new model implies the same lower bound on the number of walking steps on the original model. For any NNJAG J (modified as above) with p pebbles, q states and T time, we can construct a pebble location redundant NNJAG J' so that for any possible id (Q,Π) of J, J' will have a state $\langle Q,\Pi\rangle$. In this state, J' will perform the same action as J does on id (Q,Π).[1] Thus, the pebble location redundant NNJAG J' will have p pebbles and $q \times n^p$ states, hence using space $\log(q \times n^p) + p\log n = \log q + 2p\log n$ which is at most twice the space of J. Moreover, it uses no more time than J.

To be able to discuss subcomputations of the NNJAG better it is convenient to recast the NNJAG as an r-way branching program [BC82] (defined below). While, though, an r-way branching

[1]Note that in general, a standard NNJAG cannot be made pebble location redundant because if the move taken from an id (Q_1,Π_1) is a walk, the new pebble location, Π_2, will depend on the input graph. Hence the NNJAG cannot know which new state Q_2 to move to so that $\widehat{\Pi}(Q_2) = \Pi_2$. In contrast, in the modified NNJAG, the pebble location, Π_2, after substep 2 is uniquely determined by (Q_1,Π_1) and Π_1'. Hence it is possible for the NNJAG to choose a state Q_2 so that $\widehat{\Pi}(Q_2) = \Pi_2$.

program is a general model of computation the branching program we will examine has "structure", imposed by lemmata 6.1, 6.2 regarding NNJAG computations.

A branching program is a directed acyclic graph with a designated source node and a number of sink nodes. Each sink node in the graph is labelled with either *accept* or *reject* and each non-sink node is labelled with an input variable. Furthermore, for each possible value of the input variable that labels a non-sink node, there is a unique out-edge from this non-sink node, labelled with that value. Hence the out-degree of the graph is at most r, where r is the maximum number of different values possible for an input variable. A *sub-branching program* is simply a subgraph rooted at some node.

The nodes in this graph represent the possible states of the machine's memory. In particular, the source node represents the initial memory state. In each step the machine queries an input variable, depending on the current state of its memory, and then changes its memory to another state based on the value returned. Which variable to query and which state to go to, on each possible outcome, are specified by the graph. It is easy to see that for every input, there will be a unique path in the graph from the source node to a sink node. We call such a path the *computation path* followed by the input. We say that a branching program accepts an input if and only if the computation path followed by the input leads to a sink node labelled with accept.

Consider an arbitrary (pebble location redundant) NNJAG J that uses space S, takes time T and takes inputs from a distribution of n-node graphs with out-degree f. The corresponding branching program A has a row of configuration nodes for each of the time steps $t \in [1 \ldots T]$. Each row has 2^S configuration nodes (Q, Π, t), one for each NNJAG id (Q, Π). For every id (Q, Π) of J and time step t, there will be a configuration vertex (Q, Π, t) in A. The configuration vertex $(Q_0, \Pi_0, 1)$, where (Q_0, Π_0) is the *start* id of J, is taken as the *start* vertex of A. For each *accept* id (Q_a, Π_a) of J, $(Q_a, \Pi_a, 1), (Q_a, \Pi_a, 1), \ldots, (Q_a, \Pi_a, T)$ are *accept* configuration vertices in A. Likewise for the *reject* ids. The input variables labelling the configuration vertices of A are the variables $X_{\langle u,\ell \rangle}$ where $u \in [0..n-1]$ is a node name and $\ell \in [0..f-1]$ is an edge label. The variable $X_{\langle u,\ell \rangle}$ will have value v if there is an edge (u, v) labelled with ℓ in the input graph and the value "undefined" if there is no such edge. Thus, if in id (Q, Π) the first substep of J walks a pebble from node u along the edge with label ℓ, the configuration vertex (Q, Π, t) in A will be labelled with the variable $X_{\langle u,\ell \rangle}$. Furthermore, the vertex will have a directed edge labelled with v to configuration vertex $(Q', \Pi', t+1)$ if for some input graph, $X_{\langle u,\ell \rangle} = v$ (i.e., the queried edge has destination v) and the subsequent jumps taken in the second substep by J bring the machine to the id (Q', Π'). If J does not walk any pebble in the first substep, the configuration vertex will not get any label and will have only one unlabelled out-edge pointing to some configuration vertex $(Q', \Pi', t+1)$ depending on the second substep of J.

Note that the branching program A so constructed is *levelled* in the sense that each configuration

vertex can be assigned a level number so that edges from level i only go to level $i+1$. Moreover, all the rows in A are identical as the transition function of the (deterministic) NNJAG does not depend on time. Therefore, the number of distinct sub-branching programs of a fixed height is at most 2^S.

Finally, we introduce a variant of branching programs called *sectioned* branching programs. A branching program is said to be sectioned if its vertices are partitioned into sections so that the out-edges of a vertex in section i can only go to vertices in section i or $i+1$. Thus each computation path will go through each section at most once.

Definition 2 *A branching program A is* properly sectioned *for an input G if it queries at most $\frac{3fS}{\log n}$ different edges of G in each section. If A is properly sectioned for G then its* s-height *on G, denoted by $h_A(G)$, is $\frac{3fS}{\log n}$ times the number of sections A contains; otherwise, $h_A(G)$ is infinite.*

Note that a set of queries to the same edge of the input graph within a section is only charged as one query in the s-height measure. The branching program defined earlier can be viewed as a sectioned branching program with $T/\frac{3fS}{\log n}$ sections, each of which queries at most $\frac{3fS}{\log n}$ different input edges. Moreover, on every input G, A will have s-height $T = (T/\frac{3fS}{\log n}) \times \frac{3fS}{\log n}$.

8 Proof Outline

In the rest of this paper, a directed edge from u to v with label ℓ will be denoted by the triple $\langle u, \ell, v\rangle$. Also, by $\mathcal{G}(\mathcal{O})$ we denote the distribution obtained by selecting those graphs in $\mathcal{G}$ that satisfy a condition $\mathcal{O}$. We will derive Lemma 8.3 by induction. Before doing so, we present two lemmata that are central to the proof of that inductive statement. The first one mainly concerns traversing base edges of graphs in $\mathcal{G}_k$. It bounds the probability of a machine making a lot of progress within a short period of time.

Lemma 8.1 (Main Lemma) *Let A be any sectioned sub-branching program derived from some pebble location redundant NNJAG with at most $S/\log n$ pebbles. Then for any $k \in [1..K]$,*

$$Pr_{G\in\mathcal{G}_k}\left[\, w_A(G) \geq 3S/\log n \ \textit{and}\ h_A(G) \leq \chi/8 \,\right] < 2^{-2S}.$$

The intuition behind Lemma 8.1 is as follows. Recall that $w_A(G)$ is the number of goals that get pebbled. We "give away" one such node for each of the (at most) $S/\log n$ pebbles. When $h_A(G) \leq \chi/8$, A queries at most $\chi/8$ different edges in G. Consider the probability of pebbling the goal corresponding to an arbitrary root r, assuming that the hard path rooted at r does not contain any pebble initially. There are f^{D-1} possibilities for the vector, $\vec{\ell}_r$, of edge labels on this hard path. To remind us of its dependency on G, let us use the symbol $\vec{\ell}_r(G)$ instead of $\vec{\ell}_r$ in the following. An NNJAG can move a pebble down from r following some vector $\vec{\ell} \in [0..f-1]^{D-1}$

of edge labels, hoping that $\vec{\ell} = \vec{\ell}_r(G)$. For G drawn from $\mathcal{G}_k$, this probability is $f^{-(D-1)}$. The NNJAG can dynamically choose $\vec{\ell}$ based on the names of the nodes on the path it has traced so far. However, this will not be a lot of help as the name of the nodes on the hard path are chosen randomly. Recall that $f^D \geq n^{80}$. Since $\chi \in O(n)$, it follows that $f^{D-1} \gg \chi/8$. Clearly, by querying at most $\chi/8$ different edges in the input graph, the NNJAG cannot try many different $\vec{\ell}$s. Hence the probability of having at least one of them being successful is small.

On the other hand, the NNJAG can eliminate some of the possibilities it needs to consider, by detecting "collisions of edges" and hence increase the probability it succeeds. For example, when it learns that two different edges have the same destination node v, it learns that this node v is not on the hard path since its in-degree is bigger than one. Hence, any path continuing from node v need not be traversed. However, within $\chi/8$ steps, the probability that an edge traversed by the NNJAG collides with some other traversed edge can be shown to be at most $1/4$. (Intuitively, the probability is $\frac{\chi/8}{2^k\chi/2} \leq 1/8$. For the simplicity of the proof, we argue in Section 11 that this probabilty is at most $1/4$.) By analyzing a variant of branching processes, we can show that the probability of eliminating a large number of vectors $\vec{\ell} \in [0..f-1]^{D-1}$ in this way, is small. In other words, with high probability, the NNJAG still has a lot of possible $\vec{\ell}$s to try out. This discussion considers only a single root. When there are many roots, we need to take care of the dependencies among them before we can apply some Chernoff-type bounds. The detailed analysis and proof of Lemma 8.1 comprises Sections 10, 11 and 12.

The second lemma concerns the traversal of collapsed edges of graphs in $\mathcal{G}_k$. Let E be a fixed set of D^{k-1} base graphs $G_1, \ldots, G_{D^{k-1}} \in \mathcal{B}(2^k\chi)$ (we call such a set of graphs a *complete* set) and $\mathcal{G}_k(E)$ be the distribution of $\mathcal{G}_k$ conditioned on these fixed graphs. The lemma relates the computation of a pebble location redundant NNJAG J on inputs in $\mathcal{G}_k(E)$ to that of a faster (in terms of s-height) pebble location redundant NNJAG J' on inputs in $\mathcal{G}_{k-1}$. For any complete set E of base graphs, define a function C_E from nodes in $G \in \mathcal{G}_k(E)$ to nodes in $C(G) \in \mathcal{G}_{k-1}$ as follows:

$$C_E(v) = \begin{cases} w_i, & \text{if } v = w_i \text{ for some } i \in [1..2^{k-1}\chi] \\ u_{\langle i,j \rangle}, & \text{if } v \text{ is on the } j^{th} \text{ hard path in } G_i \\ \mathit{lost}, & \text{otherwise.} \end{cases}$$

Note that the function is well-defined because for an input $G \in \mathcal{G}_k(E)$, whether a node v is a hard node, an easy node or an auxiliary node is fixed. For any pebble mapping Π for graphs in $\mathcal{G}_k(E)$, denote by $C_E(\Pi)$ the pebble mapping Π' for graphs in $\mathcal{G}_{k-1}$ such that for any pebble P, $\Pi'(P) = C_E(\Pi(P))$.

Lemma 8.2 (Collapsing Lemma) *Let k be any integer in $[1..K]$, J be any pebble location redundant NNJAG with p pebbles and q states, and E any complete set of base graphs. There exists a corresponding pebble location redundant NNJAG J' with the same number of pebbles and states*

such that for any $G \in \mathcal{G}_k(E)$, J is in id (Q, Π) in some step on input G if and only if J' is in id $(Q, C_E(\Pi))$ in the same step on input $C(G) \in \mathcal{G}_{k-1}$.

Note that J and J' use the same space. Moreover, J traverses a collapsed edge $\langle u, \ell, v \rangle$ in G if and only if J' traverses the corresponding edge $\langle C_E(u), \ell, C_E(v) \rangle$ in $C(G)$, and J accepts G if and only if J' accepts $C(G)$. The proof of Lemma 8.2 is given in Section 9. Having stated lemmata 8.1, 8.2 we are ready to state and prove the following inductive statement.

Lemma 8.3 *For any $\epsilon > 0$ and any $k \in [0..K]$, if $T_k = \epsilon\chi\left(\frac{\chi \log n}{24fS}\right)^k$ and A is a sectioned branching program with no more than $T_k/(\frac{3fS}{\log n})$ sections, derived from a pebble location redundant NNJAG J which uses at most space S, then $Pr_{G \in \mathcal{G}_k}[\, w^*_A(G) > \epsilon\chi$ and $h_A(G) \leq T_k\,] \leq k2^{-S} < \epsilon$.*

Proof of Lemma 8.3:

(Base Case) When $k = 0$, the branching program A can query at most $T_0 = \epsilon\chi$ different edges. Hence it cannot discover more than $\epsilon\chi$ super goals.

(Inductive Step) Assume that the lemma is true for $k-1$. Consider a sectioned branching program A having at most $T_k/\frac{3fS}{\log n}$ sections derived from some pebble location redundant NNJAG J with at most S space. Suppose, for the sake of contradiction, that $\Pr_{G \in \mathcal{G}_k}[\, w^*_A(G) > \epsilon\chi$ and $h_A(G) \leq T_k\,]$ $> k2^{-S}$. We will show that in this case there exists some sectioned branching program A' with at most $T_{k-1}/\frac{3fS}{\log n}$ sections corresponding to some pebble location redundant NNJAG J' using at most S space such that $\Pr_{G' \in \mathcal{G}_{k-1}}[\, w^*_{A'}(G') > \epsilon\chi$ and $h_{A'}(G') \leq T_{k-1}\,] > (k-1)2^{-S}$. This contradicts the inductive hypothesis.

We break A into at most $T_{k-1}/\frac{3fS}{\log n}$ slices so that slice i consists of section $i(T_k/T_{k-1})$ to section $(i+1)(T_k/T_{k-1}) - 1$, inclusive. (So each slice contains $T_k/T_{k-1} = \frac{\chi \log n}{24fS}$ sections.) Let $\mathcal{F}$ be the set of $G \in \mathcal{G}_k$ such that $h_A(G) \leq T_k$ and at least one sub-branching program $\widehat{A}$, which lies completely within a slice, has $w_{\widehat{A}}(G) \geq 3S/\log n$. Since $h_A(G)$ being finite implies that $\widehat{A}$ is properly sectioned for G, $h_{\widehat{A}}(G) \leq (T_k/T_{k-1})(\frac{3fS}{\log n}) = \chi/8$.

Consider the maximal sub-branching program $\widehat{A}$ which lies completely within slice i and is rooted at the node through which G first enters slice i. There are at most 2^S such sub-branching programs in A. Combining this fact with Lemma 8.1, we have $\Pr_{G \in \mathcal{G}_k}[G \in \mathcal{F}] < 2^{-S}$. Therefore, $\Pr_{G \in \mathcal{G}_k}[\, w^*_A(G) > \epsilon\chi$ and $h_A(G) \leq T_k$ and $G \notin \mathcal{F}\,] > (k-1)2^{-S}$. Let us choose a complete set E of base graphs so that $\Pr_{G \in \mathcal{G}_k(E)}[\, w^*_A(G) > \epsilon\chi$ and $h_A(G) \leq T_k$ and $G \notin \mathcal{F}\,] > (k-1)2^{-S}$. By Lemma 8.2, we can construct from the pebble location redundant NNJAG J, another pebble location redundant NNJAG J' that runs on $\mathcal{G}_{k-1}$ with the same number of pebbles and states as J. From J', we can construct a sectioned branching program A' with at most $T_{k-1}/\frac{3fS}{\log n}$ sections; one section for each of the slices of A. This is done by putting a configuration vertex of A' in section

i if and only if the corresponding[2] configuration vertex of A is in slice i. In A, edges only go from vertices in slice i to vertices in slice i or $i+1$. Therefore, in A', edges only go from vertices in section i to vertices in section i or $i+1$. Hence, this is a legal way of partitioning the vertices of A' into sections. Now, consider an arbitrary graph $G \in \mathcal{G}_k(E) - \mathcal{F}$. At most $\frac{3S}{\log n}$ progress is made in the unique maximal sub-branching program that G passes through in each slice of A. By Lemma 6.2, each such sub-branching program can query at most $\frac{3fS}{\log n}$ different collapsed edges in G. Hence each corresponding sub-branching program in A' queries at most $\frac{3fS}{\log n}$ different edges in $C(G)$. Therefore, A' is properly sectioned for $C(G)$, for all $G \in \mathcal{G}_k(E) - \mathcal{F}$. Since A has $T_{k-1}/\frac{3fS}{\log n}$ slices, A' has the same number of sections. It follows that $h_{A'}(C(G)) \leq T_{k-1}$, for all $G \in \mathcal{G}_k(E) - \mathcal{F}$. Since $C(G)$ is chosen independent of the G_is, the distribution $\mathcal{G}_k(E)$ is isomorphic to the distribution $\mathcal{G}_{k-1}$. Therefore,

$$\begin{aligned}
&\Pr_{C(G)\in\mathcal{G}_{k-1}}[\, w^*_{A'}(C(G)) > \epsilon\chi \text{ and } h_{A'}(C(G)) \leq T_{k-1}\,] \\
&\quad \geq \Pr_{G\in\mathcal{G}_k(E)}[\, w^*_{A'}(C(G)) > \epsilon\chi \text{ and } G \notin \mathcal{F}\,] \\
&\quad = \Pr_{G\in\mathcal{G}_k(E)}[\, w^*_{A}(G) > \epsilon\chi \text{ and } G \notin \mathcal{F}\,] \\
&\quad \geq \Pr_{G\in\mathcal{G}_k(E)}[\, w^*_{A}(G) > \epsilon\chi \text{ and } h_A(G) \leq T_k \text{ and } G \notin \mathcal{F}\,] \\
&\quad > (k-1)2^{-S}.
\end{aligned}$$

■

For $k = K$ the above inductive statement implies that any deterministic pebble location redundant NNJAG which uses at most S space and takes $O(T_K)$ time, will pebble more than $\epsilon\chi$ super goals with probability less than $K2^{-S}$. Recall that $T_K = \epsilon\chi\left(\frac{\chi \log n}{24fS}\right)^K$. For $S \in O(n^{1-\delta})$, we set $f = \Theta((n\log n/S)^{1/2})$, $\chi = \Theta((n^3S/\log n)^{1/4})$ and $K = \Theta(\log(\frac{n\log n}{S}))$. Hence $T_K = 2^{\Omega(\log^2(\frac{n\log n}{S}))} \times (n^3S/\log n)^{1/4} = 2^{\Omega(\log^2(n/S))}$. For $S \in \omega(n^{1-\delta})$, we set $f = 2$, $\chi = \Theta((nS/\log n)^{1/2})$ and $K = \Theta(\log(\frac{n\log n}{S})/\log\log n)$. Thus, we get $T_K = 2^{\Omega(\log^2(\frac{n\log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$. For big enough n, $K2^{-S} < \epsilon$, since $K \leq \log n$ and $S \geq \log n$. Thus, if J is an NNJAG that uses space S and time $T \notin \Omega(T_K)$ then for any $\epsilon > 0$, $\Pr_{G\in\mathcal{D}}[\, w^*_J(G) > \epsilon\chi\,] < \epsilon$.

Note that for $S \in O(n^{1-\delta})$, the input graph has out-degree $f = \Theta((n\log n/S)^{1/2})$ which is non-constant. We can convert the graphs of out-degree f into graphs of out-degree 2 by replacing each node with a binary tree of size $O(f)$. This blows up the number of nodes by a factor of f. Hence our lower bound becomes $T \in 2^{\Omega(\log^2(n/fS))} = 2^{\Omega(\log^2(n/S))}$ where n is the number of nodes in the out-degree 2 graph.

[2]There is a 1-1 correspondence between states of J and J'. It is not hard to see that there is also a 1-1 correspondence between configuration vertices of A and A'.

9 Collapsing an NNJAG

Lemma 8.2 *Let k be any integer in $[1..K]$, J be any pebble location redundant NNJAG with p pebbles and q states, and E any complete set of base graphs. There exists a corresponding pebble location redundant NNJAG J' with the same number of pebbles and states such that for any $G \in \mathcal{G}_k(E)$, J is in id (Q, Π) in some step on input G if and only if J' is in id $(Q, C_E(\Pi))$ in the same step on input $C(G) \in \mathcal{G}_{k-1}$.*

Proof: J' will have the same set of states as J. Let $\widehat{\Pi}$ be the function that maps the states of J to its pebble locations. We shall prove, by induction on the number of steps taken, that if J is in state Q in step t on input $G \in \mathcal{G}_k(E)$ then J' is in the same state in step t on input $C(G) \in \mathcal{G}_{k-1}$ and $C_E(\widehat{\Pi}(Q))$ specifies the locations of its pebbles in that step. This proves the claim. Initially, J and J' are at state Q_0. Both $\widehat{\Pi}(Q_0)$ and $C_E(\widehat{\Pi}(Q_0))$ specify that all pebbles are on node s.

Assume that at step t, J is in state Q on input G, and at the same step, J' is in state Q on $C(G)$ and its pebble locations are specified by $C_E(\widehat{\Pi}(Q))$. The move of J' will be determined by the move of J. For the first substep there are three cases.

(Case 1) If J *does nothing* then J' also does nothing.

(Case 2) If J *walks pebble P along the edge with label ℓ* then there are 3 subcases depending on the node u that P was on.

(2a) If u is the lost node or the node w_j for some $j \in [1..2^k\chi]$ then the destination, v, of P is fixed and $C_E(v)$ is either the lost node, the node $u_{\langle 1,j\rangle}$ or w_{j+1}. Hence J' node-jumps pebble P to $C_E(v)$.

(2b) If u is a hard node in layer D of some G_i (i.e., u is a goal node) then the out-edges of u are collapsed edges. In this case, J' walks pebble P along edge ℓ.

(2c) If u is not a goal node and not an auxiliary node then the destination, v, is fixed for all $G \in \mathcal{G}_k(E)$. If $C_E(u) \neq C_E(v)$ then u must be a hard node and v must be an easy node. Hence J' node-jumps pebble P to $C_E(v)$ which is the lost node. If $C_E(u) = C_E(v)$, then J' does nothing.

(Case 3) *If J node-jumps pebble P to node v* then v must be either the lost node or w_j or $u_{\langle 1,j\rangle}$ for some $j \in [1..2^k\chi]$. Hence $C_E(v)$ is either the lost node, w_j or $u_{\langle 1,j\rangle}$ for some $j \in [1..2^{k-1}\chi]$. J' just node-jumps pebble P to $C_E(v)$.

Let Π_1 be the pebble locations of J after the first substep and let J assume state Q' in the second substep. In its second substep, J' performs the same sequence of pebble-to-pebble jumps as in the second substep of J and then assumes state Q' if and only if its pebble locations after the first substep is $C_E(\Pi_1)$.

Let us check that in all the above cases, the pebble locations of J' after the first substep is indeed $C_E(\Pi_1)$. By the inductive hypothesis pebble P of J' was on node $C_E(u)$ before the first substep while pebble P of J was on node u. Since J only moves pebble P from node u to v in the first substep, we just need to show that J' moves P from node $C_E(u)$ to $C_E(v)$ in the first substep. This is obviously true in all the above cases, except (2b). In Case (2b), $\langle u, \ell, v\rangle$ is a collapsed edge in G. By the definitions of $\mathcal{G}_k$ and $\mathcal{G}_{k-1}$, $\langle C_E(u), \ell, C_E(v)\rangle$ is an edge in $C(G)$. Hence pebble P of J' will be on node $C_E(v)$ after the first substep. It follows that J' will also assume state Q' in the second substep. Moreover, in the second substep, J changes the pebble locations from Π_1 to $\widehat{\Pi}(Q')$ by pebble-to-pebble jumps. By construction, J' will also change the pebble locations from $C_E(\Pi_1)$ to $C_E(\widehat{\Pi}(Q'))$. ∎

10 Proof of the Main Lemma

Lemma 8.1 *Let A be any sectioned sub-branching program derived from some pebble location redundant NNJAG with at most $S/\log n$ pebbles. Then for any $k \in [1..K]$,*

$$Pr_{G\in\mathcal{G}_k}\,[\,w_A(G) \geq 3S/\log n \ \textit{and}\ h_A(G) \leq \chi/8\,] < 2^{-2S}.$$

Proof: Recall that every $G \in \mathcal{G}_k$ consists of D^{k-1} graphs, G_1, G_2,..., $G_{D^{k-1}}$ chosen independently from $\mathcal{B}(2^k\chi)$, a graph G' chosen from $\mathcal{H}_{k-1}$ and some fixed auxiliary nodes and edges. Each G_i has $2^{k-1}\chi$ roots. Therefore there are $(2D)^{k-1}\chi$ roots. Recall as well that $w_A(G)$ denotes how many of the $(2D)^{k-1}\chi$ goals have been discovered and that $h_A(G)$ is a measure of the number of edges queried.

Our proof will concentrate on the traversing of the base edges in G. We assume that G' is fixed and known to A. Hence the probability is only over the graphs $G_1, \ldots, G_{D^{k-1}} \in \mathcal{B}(2^k\chi)$. Let $\mathcal{B}_k$ be the distribution $\{\,\langle G_1, \ldots, G_{D^{k-1}}\rangle \mid G_1, \ldots, G_{D^{k-1}} \in \mathcal{B}(2^k\chi)\,\}$. We allow the machine to query any variable $X_{\langle u,\ell\rangle}$ if u is a node in the top layer of a G_i. Moreover, each time a variable $X_{\langle u,\ell\rangle}$ is queried, the followings are returned: (1) the value, v, of $X_{\langle u,\ell\rangle}$, (2) whether v is a goal node, and if so, (3) its corresponding root node. With these changes, we can assume that the machine does not query any collapsed edge as there is no need.

Also, we modify A so that it has the following properties. (1) A is a decision tree (i.e. it will not forget the answer to any previous query), (2) A will not repeat any previous query, (3) each computation path γ in A queries at most $\chi/8$ different (base) edges and discovers at most $3S/\log n$ different goal nodes. (If γ queries more than $\chi/8$ different edges, we will cut it right after it queries the $(\chi/8)^{th}$ one. Similarly, if γ discovers more than $3S/\log n$ different goal nodes, we will cut it right after it discovers the $(3S/\log n)^{th}$ one.) It is clear that the modifications will not decrease the probability stated in the lemma. With all the above assumptions and modifications , we just need to show that $\Pr_{G\in\mathcal{B}_k}\,[\,w_A(G) \geq 3S/\log n\,] \leq 2^{-2S}$.

Just before A starts, each of the $S/\log n$ pebbles may already be part way down a hard path or even on a goal node. To simplify the analysis, we assume that the goals of those hard paths that contain pebbles initially, will be discovered by A. There are at most $S/\log n$ such goals. To pebble the remaining goals, we know, by Lemma 6.1, that the entire hard path must be traversed by the NNJAG. In other words, every edge in the hard path has to be queried by A. Let $w'_A(G)$ be the number of roots such that every edge on its hard path in G has been queried by A. To prove the lemma, it suffices to show that $\Pr_{G\in\mathcal{B}_k}[\, w'_A(G) \geq 2S/\log n\,] < 2^{-2S}$.

Consider an arbitrary computation path γ in A. It can be specified by the sequence of base edges, E_γ, it has queried and the sequence of node names, R_γ, specifying whether a goal node is discovered in each step (and if applicable, its corresponding root). For example, suppose γ queries the variable $X_{\langle u,\ell\rangle}$ which has the value v and then the variable $X_{\langle u',\ell'\rangle}$ which has the value v'. Suppose v is not a goal but v' is the goal node of root r then $E_\gamma = (\langle u,\ell,v\rangle, \langle u',\ell',v'\rangle)$ and $R_\gamma = (0, r)$ (assuming no root has name 0).

When γ is the computation path followed on input G we will say that "G follows γ". (It might be useful to think of G as being "processed" by A along γ.) First, let us understand what we can deduce about $\vec{\ell}_r(G)$, the sequence of edge labels on the hard path in G rooted at r, given that $G \in \mathcal{B}_k(E_\gamma)$. (Note that G may not actually follow γ as it might not agree with R_γ.) We say that a node v is a *collision node* with respect to E_γ if E_γ contains two distinct edges $\langle u,\ell,v\rangle$ and $\langle u',\ell',v\rangle$ with the same destination v. Since v has in-degree at least two, it is known to be an easy node.

In general, we can classify $\vec{\ell} \in [0..f-1]^{D-1}$ according to γ and r as follows. Suppose we trace out a path through the edges in E_γ, starting at the root r and following the sequence of edge labels $\vec{\ell}$ until the next edge to be taken is not contained in E_γ. Then one of the following three possibilities will occur.

1. The path passes through some collision node with respect to E_γ.
2. The path reaches layer D without passing through any collision node with respect to E_γ.
3. The path stops before reaching layer D and does not pass through any collision node with respect to E_γ.

We define $Y_{\langle\gamma,r\rangle}$ and $Z_{\langle\gamma,r\rangle}$ to contain the vectors $\vec{\ell} \in [0..f-1]^{D-1}$ such that when the above procedure is applied, the second and third outcomes occur, respectively.

Claim 1 *For any computation path γ in A, any input graph $G \in \mathcal{B}_k(E_\gamma)$, and any root r, $\vec{\ell}_r(G) \in Y_{\langle\gamma,r\rangle} \cup Z_{\langle\gamma,r\rangle}$.*

Proof: For any $\vec{\ell} \notin Y_{\langle\gamma,r\rangle} \cup Z_{\langle\gamma,r\rangle}$ and any input $G \in \mathcal{B}_k(E_\gamma)$, the path from the root r labelled with $\vec{\ell}$ in G contains a collision node. Since collision nodes have in-degree at least two in E_γ, they do not lie on the hard path. Therefore, $\vec{\ell}_r(G) \neq \vec{\ell}$. ∎

Definition 3 *For any computation path γ and any $G \in \mathcal{B}_k(E_\gamma)$, $Prog_{\langle\gamma,r\rangle}(G)$ is defined as the random variable indicating that all the edges in the hard path in G rooted at r are mentioned in E_γ.*

Obviously, $Prog_{\langle\gamma,r\rangle}(G)$ is true if $\vec{\ell}_r(G) \in Y_{\langle\gamma,r\rangle}$, and false if $\vec{\ell}_r(G) \in Z_{\langle\gamma,r\rangle}$. If G actually follows γ and $Prog_{\langle\gamma,r\rangle}(G)$ is true then the goal of root r is discovered. Let $y_{\langle\gamma,r\rangle} = |Y_{\langle\gamma,r\rangle}|$ and $z_{\langle\gamma,r\rangle} = |Z_{\langle\gamma,r\rangle}|$. Briefly, the probability that $Prog_{\langle\gamma,r\rangle}(G)$ is true, given that $G \in \mathcal{B}_k(E_\gamma)$ is approximately $\frac{y_{\langle\gamma,r\rangle}}{y_{\langle\gamma,r\rangle}+z_{\langle\gamma,r\rangle}}$, because all $\vec{\ell} \in Y_{\langle\gamma,r\rangle} \cup Z_{\langle\gamma,r\rangle}$ have about the same probability to be chosen as $\vec{\ell}_r(G)$.

Let $D' = D/8$. We say that root r is a *high collision root* with respect to the computation path γ if $y_{\langle\gamma,r\rangle} + z_{\langle\gamma,r\rangle} \leq f^{D'}$. Otherwise, we say that it is a *low collision root* with respect to γ. We say that γ is a *high collision computation* if there are at least $S/\log n$ high collision roots with respect to γ. Otherwise, we say that it is a *low collision computation*. Let $\mathcal{C}$ be the set of all high collision computation paths. Then

$$\begin{aligned}
&\Pr_{G\in\mathcal{B}_k}\left[w'_A(G) \geq \frac{2S}{\log n}\right] \\
&\quad\leq \sum_{\gamma\in\mathcal{C}} \Pr_{G\in\mathcal{B}_k}[G \text{ follows } \gamma] + \sum_{\gamma\notin\mathcal{C}} \Pr_{G\in\mathcal{B}_k}\left[w'_A(G) \geq \frac{2S}{\log n} \text{ and } G \text{ follows } \gamma\right] \\
&\quad\equiv SUM_1 + SUM_2 \ .
\end{aligned}$$

By Claim 2 in Section 11, SUM_1 is at most 2^{-3S}. Consider SUM_2. If both events "$w'_A(G) \geq 2S/\log n$" and "G follows γ" occur, there exist at least $2S/\log n$ roots r such that $Prog_{\langle\gamma,r\rangle}(G)$ is true, i.e., all the edges on the hard path rooted at r in G are in E_γ. For $\gamma \notin \mathcal{C}$, at least $S/\log n$ of these are low collision roots with respect to E_γ.

Definition 4 *For any computation path γ and any $G \in \mathcal{B}_k(E_\gamma)$, $w''_\gamma(G)$ is defined as the number of roots r such that r is a low collision root with respect to γ and $Prog_{\langle\gamma,r\rangle}(G)$ is true.*

Then,

$$SUM_2 \leq \sum_{\gamma\notin\mathcal{C}} \Pr_{G\in\mathcal{B}_k}\left[w''_\gamma(G) \geq \frac{S}{\log n} \text{ and } G \text{ follows } \gamma\right].$$

Since "G follows γ" implies "$G \in \mathcal{B}_k(E_\gamma)$",

$$\begin{aligned}
SUM_2 &\leq \sum_{\gamma\notin\mathcal{C}} \Pr_{G\in\mathcal{B}_k}\left[w''_\gamma(G) \geq \frac{S}{\log n} \text{ and } G \in \mathcal{B}_k(E_\gamma)\right] \\
&\leq \max_{\gamma\notin\mathcal{C}} \Pr_{G\in\mathcal{B}_k}\left[w''_\gamma(G) \geq \frac{S}{\log n} \mid G \in \mathcal{B}_k(E_\gamma)\right] \times \sum_{\gamma\notin\mathcal{C}} \Pr_{G\in\mathcal{B}_k}[G \in \mathcal{B}_k(E_\gamma)].
\end{aligned}$$

We claim that for each graph $G \in \mathcal{B}_k$, there are at most 2^{6S} different computation paths γ for which G satisfies E_γ. To see this, observe that every computation path γ in A queries at most $\chi/8$ different base edges and discovers at most $3S/\log n$ different goal nodes, each having at most n name choices for its corresponding root. Hence there are at most $\binom{\chi/8}{3S/\log n} \times n^{3S/\log n} \leq 2^{6S}$

different sequences R_γ. If there were more than 2^{6S} different computation paths γs such that G satisfies E_γ, then there exists two different computation paths γ and γ' such that $R_\gamma = R_{\gamma'}$ and G satisfies both E_γ and $E_{\gamma'}$. For γ and γ' to be different, there must be an edge $\langle u, \ell, v\rangle$ in E_γ and an edge in $\langle u, \ell, v'\rangle$ in $E_{\gamma'}$ such that $v \neq v'$. Then G cannot satisfy both E_γ and $E_{\gamma'}$, a contradiction. Hence our claim follows. From this claim, we have $\sum_{\gamma \notin \mathcal{C}} \Pr_{G \in \mathcal{B}_k}[G \in \mathcal{B}_k(E_\gamma)] \leq 2^{6S}$ and thus,

$$SUM_2 \leq \max_{\gamma \notin \mathcal{C}} \Pr_{G \in \mathcal{B}_k}\left[w''_\gamma(G) \geq \frac{S}{\log n} \mid G \in \mathcal{B}_k(E_\gamma) \right] \times 2^{6S}$$

Claim 4 of Section 12 shows that $\Pr_{G \in \mathcal{B}_k}\left[w''_\gamma(G) \geq S/\log n \mid G \in \mathcal{B}_k(E_\gamma) \right]$ is at most 2^{-9S} for any γ in A. In conclusion,

$$\begin{aligned} SUM_1 + SUM_2 &\leq 2^{-3S} + 2^{-9S+6S} \\ &\leq 2^{-3S+1} \\ &\leq 2^{-2S}, \end{aligned}$$

where all inequalities hold for big enough n. Hence Lemma 8.1 (Main Lemma) follows. ∎

11 Bounding SUM_1

This section bounds the first sum, SUM_1, at the end of the proof for Lemma 8.1 (Main Lemma).

Claim 2 $\sum_{\gamma \in \mathcal{C}} Pr_{G \in \mathcal{B}_k}[G \textit{ follows } \gamma] \leq 2^{-3S}$.

Proof: We first define two games called the *edge-collision game* and the *branching-process game*. Let S_{ed} and S_{br} be the random variables indicating the success of each game, respectively. We shall show that $\sum_{\gamma \in \mathcal{C}} \Pr_{G \in \mathcal{B}_k}[G \text{ follows } \gamma] \leq \Pr[S_{ed}] \leq \Pr[S_{br}] \leq 2^{-3S}$. ∎

11.1 The Edge-Collision Game

The *edge-collision game* is defined as follows. D^{k-1} graphs $G_1, G_2, \ldots, G_{D^{k-1}}$ are chosen randomly and independently from $\mathcal{B}(2^k\chi)$. The player is informed of the hard path of each root in each G_i. He then queries edges of the G_is one at a time. When the player queries an edge, he specifies $\langle u, \ell\rangle$, where u is a node and $\ell \in [0..f-1]$ is an edge label. The destination node v of the edge $\langle u, \ell, v\rangle$ is then revealed to the player. Based on the result of the previous queries, he chooses the next edge to query. He is allowed to query at most $\chi/8$ edges in total.

The aim of the player is to minimize the number of leaves of certain trees associated with the queried edges. To be precise, let E be the sequence of base edges of the input graph G that the player has queried during the game. Recall that in Section 10 a node v is called a *collision node*

with respect to E if it is the destination of more than one edge in E. Here in this game, each edge in E will be in one of two conditions: *alive* or *dead*. An edge is said to be dead if its destination node is (1) a collision node or (2) the source of a previously queried edge. Otherwise, it is alive. We shall construct from E a collection of f-ary trees by taking the following steps.

(Step 1) If E does not contain a path from any root r to a node v then delete v from G (along with all its in/out-edges),

(Step 2) Delete all the nodes (along with their in/out-edges) that are proper descendents in E of the destinations of the dead edges. The dead edges and their destinations are kept. The remaining edges in E that are alive are called the *y-edges* and their destination nodes are called the *y-nodes*. Each such node has a unique path from some root to it and that path contains no dead edges. Hence, the y-edges form a collection of disjoint f-ary trees.

(Step 3) "Fill up" the above trees so that each node has exactly f outgoing edges. More precisely, for each y-node that does not have exactly f outgoing edges (counting the dead edges), add the missing edges and attach to each such edge a complete f-ary tree, of appropriate depth, such that its leaves are at layer D. The nodes and edges that are added in this way are referred to as the *z-nodes* and the *z-edges*. They do not correspond to actual nodes and edges in the input graph G. Note that each z-node also has a unique path from some root to it and that path contains no dead edges.

We shall measure the performance of the game player by two sets of parameters. They are somewhat similar to $y_{\langle\gamma,r\rangle}$, $z_{\langle\gamma,r\rangle}$ defined in Section 10. Define $\tilde{y}_r$ and $\tilde{z}_r$ (the performance parameters) as the number of y and z-nodes at layer D, that are descendents of the root node r. Again, a root r is said to be a *high collision* root if $\tilde{y}_r + \tilde{z}_r \leq f^{D'}$, where $D' = D/8$ as defined in Section 10. The goal of the player is to create as many high collision roots as possible. More precisely, the player wins if there are more than $S/\log n$ high collision roots. Let S_{ed} be the indicator variable of this event.

Note that the edge-collision game player can query whatever edges queried by A, and hence ensure that $E_\gamma \subseteq E$ where E_γ and E are the set of edges queried by A and the game player respectively. Consider the conditions for a root to be a high collision one. In A, a root r is a high collision root with respect to E_γ if $y_{\langle\gamma,r\rangle} + z_{\langle\gamma,r\rangle} \leq f^{D'}$, i.e., the number of vectors $\vec{\ell} \in Y_{\langle\gamma,r\rangle} \cup Z_{\langle\gamma,r\rangle}$ is small. If a vector $\vec{\ell} \in [0..f-1]^{D-1}$ is not in $Y_{\langle\gamma,r\rangle} \cup Z_{\langle\gamma,r\rangle}$, then the path obtained by following $\vec{\ell}$ from r must contain a collision node in E_γ. The same node will also be a collision node in the edge-collision game provided $E_\gamma \subseteq E$. It follows that a high collision root in the A will also be a high collision root in the game. Therefore, $\sum_{\gamma\in\mathcal{C}} \Pr_{G\in\mathcal{B}_k}[G \text{ follows } \gamma] \leq \Pr[S_{ed}]$.

11.2 The Branching-Process Game

Before we introduce the branching process game we introduce some machinery that will be useful in bounding its probability of success.

Consider a rooted, complete, f-ary tree of depth d. We allow every edge of such a tree to *die* independently of all other edges with a fixed probability α. A node u is said to be *alive* if and only if no edge along the unique path from the root to u is dead. If Z_i denotes the number of alive vertices at level i then the sequence $Z_0 = 1, Z_1, \ldots, Z_i, \ldots$ forms a branching process [AN72]. We will be interested in the distribution of the number of alive vertices with depth d, i.e. the random variable Z_d. The expected number of alive children for an alive node is $(1-\alpha)f$ and the expected value of Z_d is $((1-\alpha)f)^d$. More precisely, the generating function for the offspring distribution in this branching process is $g(x) = (\alpha + (1-\alpha)x)^f$ (i.e., the probability that a node has i out-edges that do not die is the coefficient of x^i in $g(x)$). A well-known fact is that if $(1-\alpha)f > 1$ then $\Pr[Z_d = 0] = \xi$, where ξ is the unique $x \in (0,1)$ such that $g(x) = x$. Moreover $\eta = g'(\xi) < 1$. The following lemma states that the probability that Z_d is much smaller than its expected value is not much greater than the probability it is zero.

Lemma 11.1 [Pip92] *If $(1-\alpha)f > 1$ then for $\tilde{d} \in [1, \ldots, d]$ such that $d - \tilde{d} \to \infty$ and $d \to \infty$,*

$$Pr\left[Z_d \leq ((1-\alpha)f)^{\tilde{d}}\right] \leq \xi + O(\eta^{d-\tilde{d}}).$$

In the branching process game we will consider a variant of the above trees, as defined in Section 11.1, with depth D and out-degree f. For these trees, the variation lies in the existence of a *fixed* path from the root to some leaf whose edges are guaranteed to survive; all other edges die independently with probability 1/4. Such a tree is said to *wither* if it has at most $f^{D'}$ alive leaves. We want to bound the probability, ρ, that this happens.

To allow for a uniform treatment we first convert the f-ary trees to binary ones when $f > 2$. In particular, if $f > 2$ then $f = O(n^\epsilon)$, for some $\epsilon > 0$, and hence we can assume that f is a power of 2. Thus, we replace each node v and its f edges/children with a complete binary tree T_v of depth $\log f$, i.e. with f leaves. If v is not on the path that is guaranteed to survive then all the edges in T_v die independently, with probability 1/4. Otherwise, the edges along a unique path of T_v (the one corresponding to the edge guaranteed to live) are guaranteed to survive while the rest die independently with probability 1/4. It is easy to see, inductively, that for any set of nodes at depth $i \log f$, $i = 0 \ldots D$, in the resulting binary tree, the probability of being alive is no more than that of the corresponding nodes at depth i in the f-ary tree. Moreover, $D = \lceil 80 \log n \rceil / \log f$, for all f, and hence it will suffice to prove the bound for $f = 2$ ($D = \lceil 80 \log n \rceil$.)

Let v_i be the node at depth i which is on the path whose edges are guaranteed to survive and let v_i' be the sibling of v_i. Let ρ' be the probability that for all v_i, $i \in [1..2D/3]$, either v_i' is dead or v_i' is alive but the subtree rooted at v_i', $T_{v_i'}$, has at most $f^{D'}$ alive nodes, at depth D. Clearly, $\rho \leq \rho'$. Each $T_{v_i'}$, is a complete binary tree of depth $D-i$ where every edge dies with probability $1/4$. For $\alpha = 1/4$ and $f = 2$, we have $\xi = 1/9$ and $\eta = 1/2$. As D tends to infinity with n and $T_{v_i'}$ has depth at least $D/3$, we can apply Lemma 11.1 to bound the probability that $T_{v_i'}$ has fewer than $f^{D'}$ alive nodes at depth D (given that v_i' is alive) by $1/9 + O((1/2)^{D-i-D'})$. Thus, the probability that $T_{v_i'}$ has at most $f^{D'}$ alive leaves at depth D is no more than $1/4 + 1/9 + O((1/2)^{D-i-D'}) \leq 1/4 + 1/9 + O((1/2)^{D/3-D'}) = \beta < 1/2$, since $D' = D/8$, $D = \lceil 80 \log n \rceil$ and n can be arbitrarily large. Since the $2D/3$ subtrees grow independently, we get $\rho \leq \rho' \leq \beta^{2D/3} \in O(n^{-5})$.

In the branching process game in our graph we say that the root r of a tree as above is a *high collision* root if the tree rooted at r withers. Since there are at most n roots the expected number of high collision roots is $\mu \leq n \times \rho \in O(n^{-4})$. Let S_{br} be the random variable indicating the event that there are more than $S/\log n$ high collision roots. As each tree grows independently of the others, we can apply the Chernoff bound and prove that

$$\begin{aligned} & \Pr[S_{br}] \\ &= \Pr\left[\text{ number of high collision roots } \geq \left(\frac{S}{\mu \log n}\right)\mu\right] \\ &\leq 2^{-(\frac{S}{\log n})(\log(\frac{S}{\mu \log n}) - \log e)}. \end{aligned}$$

Since $\mu \in O(n^{-4})$, $\Pr[S_{br}] \leq 2^{-3S}$.

11.3 Branching-Process Game vs Edge-Collision Game

This subsection proves the second inequality mentioned in the proof of Claim 2.

Lemma 11.2 *The success of the edge-collision game is probabilistically dominated by the success of the branching processes game, i.e.* $Pr[S_{ed}] \leq Pr[S_{br}]$.

Proof: The edge-collision game starts by randomly choosing the graphs $G_1, G_2, \ldots, G_{D^{k-1}}$ in $\mathcal{B}(2^k\chi)$. For each $i \in [1..D^{k-1}]$, the first step in choosing G_i according to the distribution $\mathcal{B}(2^k\chi)$ is to randomly partition the $2^k\chi$ nodes at each layer into $2^k\chi/2$ easy nodes and $2^k\chi/2$ hard nodes and to choose the hard path rooted at each root r. This information is revealed to the player. The edges in these paths correspond to the edges in the branching process game that are guaranteed to live.

The next step in choosing G_i according to the distribution $\mathcal{B}(2^k\chi)$ is to choose for every remaining edge its destination among the $2^k\chi/2$ easy nodes at the next layer. This only needs to be done for those edges queried by the player. For each $i \in [1..D^{k-1}]$, for each layer $d \in [1..D]$, and for each $\tau \in [1..\frac{\chi}{8}]$, define the variable $v_{\langle i,d,\tau\rangle}$ to uniformly and independently take on a value from

$[1..\frac{2^k\chi}{2}]$. Suppose that the player is querying $\langle u, \ell \rangle$ where u is a node at layer d in the graph G_i and this is the τ^{th} query to easy edges at this layer in this graph. Then the variable $v_{\langle i,d,\tau \rangle} \in [1..\frac{2^k\chi}{2}]$ specifies the other endpoint of edge $\langle u, \ell \rangle$ among the $2^k\chi/2$ easy nodes at the next layer.

Consider the edge $\langle u, \ell \rangle$ queried at time t for some $t \in [1..\chi/8]$. Before this query, we do not know in advance which edges will be queried after time t because the player is able to choose them dynamically based on the result of the current one. However, the random variables $v_{\langle i,d,\tau \rangle}$ do tell us the resulting destinations of all the edges queried or to be queried. Together with the knowledge of the source of edges queried before time t, we can tell whether the current edge will die. Specifically, suppose $\langle u, \ell \rangle$ is the τ^{th} edge queried at layer d in G_i. Then it will die if either (1) there is another query at this layer $\tau' \in [1..\frac{\chi}{8}]$ with the same destination, i.e. $v_{\langle i,d,\tau \rangle} = v_{\langle i,d,\tau' \rangle}$ for some $\tau' \neq \tau$ or (2) the destination is the source of some edge queried before time t.

In order to compare the success probability of the edge-collision game and the branching-process game, let us first define random variables that will indicate which edges die in the branching-process game. For each root r in each of the graphs G_i, there is a corresponding root in the branching-process game. Consider the complete f-ary tree of height D rooted at such a root r. A specific edge in this tree can be specified by a string $\vec{\ell} \in [0..f-1]^*$. For each such edge, let $x_{\langle r,\vec{\ell} \rangle} \in \{0,1\}$ be the random variable indicating whether this edge dies. If the edge is one of the edges that are guaranteed to live in the branching-process game, i.e., the fixed hard path in the input graph, then $\Pr\left[x_{\langle r,\vec{\ell} \rangle}\right] = 0$. Otherwise, $\Pr\left[x_{\langle r,\vec{\ell} \rangle}\right] = 1/4$ independent of the other x variables.

Now consider a fixed algorithm for the edge-collision game. For each intermediate time step $t \in [0..\chi/8]$, we define the t^{th} game as follows. The game starts with t time steps of the fixed algorithm for the edge-collision game. Let E_t be the resulting base edges queried. We want these edges to die in the t^{th} game if and only if they die in the edge-collision game. As previously mentioned, the edge associated with $v_{\langle i,d,\tau \rangle}$ is dead if and only if (1) there is another query at this layer $\tau' \in [1..\frac{\chi}{8}]$ with the same destination, i.e. $v_{\langle i,d,\tau \rangle} = v_{\langle i,d,\tau' \rangle}$, or (2) the destination, i.e. $v_{\langle i,d,\tau \rangle}$, is the source of an edge queried before time t. In (1), the query associated with $v_{\langle i,d,\tau' \rangle}$ can occur either before or after time step t. Either way, we consider the edge associated with $v_{\langle i,d,\tau \rangle}$ dead. Given which edges in E_t have died, the set E_t can be transformed, as described in Steps (1) to (3) in Section 11.1, into a collection of f-ary trees made up of y-nodes and y-edges (the living ones), z-nodes and z-edges as well as some collision nodes and dead edges. The t^{th} game is completed by completing the branching process on the z-edges. Namely, each such edge will live or die according to the corresponding random variable $x_{\langle r,\vec{\ell} \rangle} \in \{0,1\}$. A node u in the resulting collection of trees is said to be *alive* if all the edges on the path from the root of the tree to u are alive. A root is said to be a *high collision* root if it has at most $f^{D'}$ living nodes in layer D. The t^{th} game succeeds if there are more than $S/\log n$ high collision roots. Let S_t be the random variable indicating the success of the game.

Observe that the 0^{th} game is simply the branching-process game and hence $S_{br} = S_0$. The $(\chi/8)^{th}$ game differs from the edge-collision game only in that in the edge-collision game all the z-nodes/edges added in Step 3 are treated as alive while in the $(\chi/8)^{th}$ game some of the z-edges may die according to the $x_{\langle r,\vec{\ell}\rangle}$ variables. The additional children at layer D only hurt the edge-collision game player. Therefore, $\Pr\left[S_{(\chi/8)}\right] \geq \Pr\left[S_{ed}\right]$. What remains to prove is that for every $t \in [1..\chi/8]$, $\Pr\left[S_{t-1}\right] \geq \Pr\left[S_t\right]$.

Let $\vec{V}_{(<t)}$ specify a possible computation up to and including the $(t-1)^{st}$ query. It will specify the values of $t-1$ of the $v_{\langle i,d,\tau\rangle}$ variables. Which of them are specified will depend dynamically on the computation. The computation $\vec{V}_{(<t)}$ will also specify the set of queried edges in the graph E_{t-1} and the next query $\langle u,\ell\rangle$ made by the player. Let the node u be on layer d of G_i and the query be the τ^{th} one at this layer in this graph.

Let us consider the following cases. In the first case, E_{t-1} does not contain a unique path with no dead edges from a root to u. In this case, the descendant nodes and edges of node u will be deleted from the y-node tree, both in the $(t-1)^{st}$ game and in the t^{th} game. Hence, whether this edge dies has no effect on either game. In the second case, $\langle u,\ell\rangle$ is on a hard path and for both games, the edge is guaranteed to live.

In the third case, E_{t-1} contains a unique path with no dead edges from a root to u and $\langle u,\ell\rangle$ is not on a hard path. Let r and $\vec{\ell}$ specify the root and the labels in this path. In the $(t-1)^{st}$ game, whether the edge from $\langle u,\ell\rangle$ dies is specified by the variable $x_{\langle r,\vec{\ell}\rangle} \in \{0,1\}$. In the t^{th} game, the destination of the edge from $\langle u,\ell\rangle$ is specified by the variable $v_{\langle i,d,\tau\rangle}$. Consider one setting $\vec{V}_{(>t)}$ of all the $v_{\langle i',d',\tau'\rangle}$ variables other than those set by $\vec{V}_{(<t)}$ and other than the variable $v_{\langle i,d,\tau\rangle}$. Consider as well one setting $\vec{X}_{(\neq t)}$ of all the x variables other than $x_{\langle r,\vec{\ell}\rangle}$.

Compare $\Pr\left[S_{t-1} \;\middle|\; \vec{V}_{(<t)}, \vec{V}_{(>t)}, \vec{X}_{(\neq t)}\right]$ and $\Pr\left[S_t \;\middle|\; \vec{V}_{(<t)}, \vec{V}_{(>t)}, \vec{X}_{(\neq t)}\right]$. In both cases, the probability is only over the values of $v_{\langle i,d,\tau\rangle}$ and $x_{\langle r,\vec{\ell}\rangle}$. Everything else is fixed by $\vec{V}_{(<t)}$, $\vec{V}_{(>t)}$ and $\vec{X}_{(\neq t)}$. For every value of $v_{\langle i,d,\tau\rangle}$ and $x_{\langle r,\vec{\ell}\rangle}$, which edges die before time step t and which die after time step t is the same for both the $(t-1)^{st}$ and the t^{th} game. The only change in the game is whether or not the edge from $\langle u,\ell\rangle$ dies. In the $(t-1)^{st}$ game, this edge dies with probability $\Pr\left[x_{\langle r,\vec{\ell}\rangle} \;\middle|\; \vec{V}_{(<t)}, \vec{V}_{(>t)}, \vec{X}_{(\neq t)}\right] = 1/4$. In the t^{th} game, this edge dies if there exists a $\tau' \in [1..\frac{\chi}{8}]$ $(\tau' \neq \tau)$ for which $v_{\langle i,d,\tau\rangle} = v_{\langle i,d,\tau'\rangle}$ or $v_{\langle i,d,\tau\rangle}$ is equal to the source of an edge queried before time t. $\vec{V}_{(<t)}$ and $\vec{V}_{(>t)}$ fix at most $\frac{\chi}{8}-1$ different values of the variables $v_{\langle i,d,\tau'\rangle}$ and at most $t-1 \leq \frac{\chi}{8}-1$ different values as the sources. The value for $v_{\langle i,d,\tau\rangle}$ is chosen uniformly from $[1..\frac{2^k\chi}{2}]$. Therefore, the probability that $v_{\langle i,d,\tau\rangle}$ collides with one of these $\leq 2(\frac{\chi}{8}-1)$ values given $\vec{V}_{(<t)}$, $\vec{V}_{(>t)}$, $\vec{X}_{(\neq t)}$ is at most $1/4$. Having a smaller probability of this edge dying can only hurt the t^{th} game player. We can conclude that

$$\Pr\left[S_{t-1} \;\middle|\; \vec{V}_{(<t)}, \vec{V}_{(>t)}, \vec{X}_{(\neq t)}\right] \geq \Pr\left[S_t \;\middle|\; \vec{V}_{(<t)}, \vec{V}_{(>t)}, \vec{X}_{(\neq t)}\right]$$

and hence

$$\begin{aligned}
&\Pr[S_{t-1}] \\
&= \sum_{\vec{V}_{(<t)},\vec{V}_{(>t)},\vec{X}_{(\neq t)}} \Pr\left[S_{t-1} \;\middle|\; \vec{V}_{(<t)},\vec{V}_{(>t)},\vec{X}_{(\neq t)}\right] \times \Pr\left[\vec{V}_{(<t)},\vec{V}_{(>t)},\vec{X}_{(\neq t)}\right] \\
&\geq \sum_{\vec{V}_{(<t)},\vec{V}_{(>t)},\vec{X}_{(\neq t)}} \Pr\left[S_{t} \;\middle|\; \vec{V}_{(<t)},\vec{V}_{(>t)},\vec{X}_{(\neq t)}\right] \times \Pr\left[\vec{V}_{(<t)},\vec{V}_{(>t)},\vec{X}_{(\neq t)}\right] \\
&= \Pr[S_t]. \ \blacksquare
\end{aligned}$$

12 Bounding SUM_2

This section bounds the second sum at the end of the proof of Lemma 8.1. It suffices to show that $\Pr_{G\in\mathcal{B}_k}\left[\, w''_\gamma(G) \geq S/\log n \mid G \in \mathcal{B}_k(E_\gamma)\,\right] \leq 2^{-9S}$. The event $w''_\gamma(G) \geq S/\log n$ happens when at least $S/\log n$ of the low collision roots r have $Prog_{\langle\gamma,r\rangle}(G)$ true. If for every root r, $Prog_{\langle\gamma,r\rangle}(G)$ were true with a fixed probability independent of the other roots, then we could apply the Chernoff bound directly. However, there are indeed dependencies among different roots. Fortunately if each event has a low probability of success no matter what outcomes of the other events have, then by the following lemma from Edmonds [Edm93a] the Chernoff bound still holds.

Lemma 12.1 (Lemma 14 of [Edm93a]) *Let $\mathcal{R}$ be the set of roots. For each $r \in \mathcal{R}$, let $\hat{x}_r \in \{0,1\}$ be the random variable indicating the success of the r^{th} trial. For each $r \in \mathcal{R}$ and $\mathcal{O} \in \{0,1\}^{\mathcal{R}-\{r\}}$, let $Z_{\langle r,\mathcal{O}\rangle} = Pr[\hat{x}_r = 1 \mid \mathcal{O}]$, where $\mathcal{O}$ indicates that the other trials have the stated outcomes. If for every r and every possible outcome of the other trials $\mathcal{O}$, $Z_{\langle r,\mathcal{O}\rangle} \leq \rho$, then for every $\delta > 1$, $Pr[\sum_{r\in\mathcal{R}} \hat{x}_r \geq 2\delta\rho|\mathcal{R}|] \leq 2^{-0.38\delta\rho|\mathcal{R}|}$.*

Proof: Let $\hat{X} = \sum_{r\in\mathcal{R}} \hat{x}_r$. To bound $\Pr\left[\hat{X} \geq 2\delta\rho|\mathcal{R}|\right]$, we will consider a sequence of random variables, x_r, $r \in \mathcal{R}$, defined as follows: for x_1, we choose uniformly at random $\lambda_1 \in [0,1]$ and set $x_1 = 1$ iff $\lambda_1 \leq \Pr[\hat{x}_1 = 1]$. In general, if we have set $x_1 = a_1, \ldots, x_i = a_i$, we choose uniformly at random $\lambda_{i+1} \in [0,1]$ and set $x_{i+1} = 1$ iff $\lambda_{i+1} \leq \Pr[\hat{x}_{i+1} = 1 \mid \hat{x}_1 = a_1 \wedge \cdots \wedge \hat{x}_i = a_i]$. Clearly, the sequences $\hat{x}_r$ and x_r are identically distributed and $\Pr[x_r = 1] \leq \rho$, for all $r \in \mathcal{R}$.

Consider now a sequence of random variables y_r defined by $y_r = 1$ iff $\lambda_r \leq \rho$, $r \in \mathcal{R}$ where λ_r is as above. By construction, $x_r \leq y_r$ for all $r \in \mathcal{R}$. Hence, if $X = \sum_{r\in\mathcal{R}} x_r$ and $Y = \sum_{r\in\mathcal{R}} y_r$, then $X \leq Y$. Moreover Y is the sum of $|R|$ independent Boolean random variables. Applying the Chernoff bound we get that for $\delta > 1$,

$$\Pr\left[\hat{X} \geq 2\delta\rho|\mathcal{R}|\right] = \Pr[X \geq 2\delta\rho|\mathcal{R}|] \leq \Pr[Y \geq 2\delta\rho|\mathcal{R}|] \leq 2^{-0.38\delta\rho|\mathcal{R}|} \ .$$

$\blacksquare$

In Claim 3, we first show that the probability of $Prog_{\langle\gamma,r\rangle}(G)$ being true is small for low collision roots r. Then we will apply the above lemma in Claim 4 to get the desired bound for the second sum.

Claim 3 *For any computation path γ in A, any root r and any subset $\mathcal{O}$ of roots indicating for which roots r' other than r, $Prog_{\langle\gamma,r'\rangle}(G)$ is true,*

$$Pr_{G\in\mathcal{B}_k}\left[\, Prog_{\langle\gamma,r\rangle}(G) \mid G \in \mathcal{B}_k(E_\gamma) \text{ and } \mathcal{O}\,\right] \;\leq\; \frac{4}{3}\frac{y_{\langle\gamma,r\rangle}}{y_{\langle\gamma,r\rangle}+z_{\langle\gamma,r\rangle}}$$

Proof: Let us consider a fixed γ and r. Recall that $\vec{\ell}_r(G)$ is the random variable indicating the vector of edge labels on the hard path rooted at r in graph G drawn from $\mathcal{B}_k$ and that $\vec{\ell}_r(G) \in Y_{\langle\gamma,r\rangle} \cup Z_{\langle\gamma,r\rangle}$. Recall as well that $Prog_{\langle\gamma,r\rangle}(G)$ is true if and only if $\vec{\ell}_r(G) \in Y_{\langle\gamma,r\rangle}$. We shall drop the subscripts in $Y_{\langle\gamma,r\rangle}$, $Z_{\langle\gamma,r\rangle}$, $y_{\langle\gamma,r\rangle}$, $z_{\langle\gamma,r\rangle}$, $\vec{\ell}_r(G)$, $Prog_{\langle\gamma,r\rangle}(G)$ and E_γ when there is no confusion. We shall also write $\Pr_{G\in\mathcal{B}_k}[\,\cdot \mid G \in \mathcal{B}_k(E_\gamma)\,]$ as $\Pr[\,\cdot \mid E\,]$. Note that

$$\begin{aligned}
&\Pr[\,Prog(G) \mid E \text{ and } \mathcal{O}\,] \\
&= \frac{\Pr[\,Prog(G) \mid E \text{ and } \mathcal{O}\,]}{\Pr[\,Prog(G) \mid E \text{ and } \mathcal{O}\,] + \Pr[\,\neg Prog(G) \mid E \text{ and } \mathcal{O}\,]} \\
&= \frac{\sum_{\vec{\ell}\in Y}\Pr\left[\,\vec{\ell}(G)=\vec{\ell}\mid E \text{ and } \mathcal{O}\,\right]}{\sum_{\vec{\ell}\in Y}\Pr\left[\,\vec{\ell}(G)=\vec{\ell}\mid E \text{ and } \mathcal{O}\,\right] + \sum_{\vec{\ell}\in Z}\Pr\left[\,\vec{\ell}(G)=\vec{\ell}\mid E \text{ and } \mathcal{O}\,\right]}
\end{aligned}$$

Let $\vec{\ell}_y$ be the vector in Y that maximizes $\Pr\left[\,\vec{\ell}(G)=\vec{\ell}\mid E \text{ and } \mathcal{O}\,\right]$ over $\vec{\ell}\in Y$ and let $\vec{\ell}_z$ be the vector in Z that minimizes $\Pr\left[\,\vec{\ell}(G)=\vec{\ell}\mid E \text{ and } \mathcal{O}\,\right]$ over $\vec{\ell}\in Z$. The above probability is at most

$$\begin{aligned}
&\frac{y \times \Pr\left[\,\vec{\ell}(G)=\vec{\ell}_y \mid E \text{ and } \mathcal{O}\,\right]}{y \times \Pr\left[\,\vec{\ell}(G)=\vec{\ell}_y \mid E \text{ and } \mathcal{O}\,\right] + z \times \Pr\left[\,\vec{\ell}(G)=\vec{\ell}_z \mid E \text{ and } \mathcal{O}\,\right]} \\
&\qquad = \frac{y}{y + \frac{\Pr[\,\vec{\ell}(G)=\vec{\ell}_z \mid E \text{ and } \mathcal{O}\,]}{\Pr[\,\vec{\ell}(G)=\vec{\ell}_y \mid E \text{ and } \mathcal{O}\,]} \times z}
\end{aligned}$$

What remains to be proven is that

$$\frac{\Pr\left[\,\vec{\ell}(G)=\vec{\ell}_z \mid E \text{ and } \mathcal{O}\,\right]}{\Pr\left[\,\vec{\ell}(G)=\vec{\ell}_y \mid E \text{ and } \mathcal{O}\,\right]} \;\geq \frac{3}{4}$$

Let $N_y(G)$ and $N_z(G)$ be respectively the set of edges on the path with label $\vec{\ell}_y$ and $\vec{\ell}_z$ from root r in G. Let $H(G)$ be the random variable specifying the hard path rooted at r in G, i.e. both the nodes and the labels $\vec{\ell}(G)$. The fact that $\vec{\ell}_y \in Y$ means that the path following the edge labels in $\vec{\ell}_y$ is totally contained in E. Therefore, $N_y(G)$ is equal to some fixed value N_y determined by E. Then the statements $\vec{\ell}(G)=\vec{\ell}_y$ and $\vec{\ell}(G)=\vec{\ell}_z$ are equivalent to $H(G)=N_y$ and $H(G)=N_z(G)$

respectively. The possible values N_z for the random variable $N_z(G)$ (i.e. the path in G rooted at r with edge labels $\vec{\ell}_z$) can be divided into two sets. Let $N_z \in \mathcal{A}_z$ if and only if some edge $\langle u, \ell, v\rangle$ in N_z has the same destination with a different edge $\langle u', \ell', v'\rangle$ in E, i.e., $v = v'$ but $\langle u, \ell\rangle \neq \langle u', \ell'\rangle$. In this case, N_z cannot be the hard path. That is, for $N_z \in \mathcal{A}_z$, $\Pr[H(G) = N_z \mid E \text{ and } \mathcal{O}] = 0$. Now consider an $N_z \notin \mathcal{A}_z$. Given that G contains $E \cup N_z$ and satisfies $\mathcal{O}$, we argue that it is equally likely for $H(G)$ to be N_y or N_z. To see this, first observe that $\mathcal{O}$ does not affect how $H(G)$ can be chosen because $\mathcal{O}$ is a condition on the hard paths of roots other than r. Secondly, both fixed paths N_y and N_z are contained in $E \cup N_z$, started from root r. Furthermore, neither N_y nor N_z contains any collision node with respect to $E \cup N_z$. By symmetry, it is equally likely for N_y and N_z to be chosen as $H(G)$. Hence,

$$\begin{aligned}
&\Pr[H(G) = N_y \mid N_z(G) = N_z \text{ and } E \text{ and } \mathcal{O}] \\
&= \Pr[H(G) = N_z \mid N_z(G) = N_z \text{ and } E \text{ and } \mathcal{O}]
\end{aligned}$$

Note as well that "$H(G) = N_z$" implies "$N_z(G) = N_z$". Therefore,

$$\begin{aligned}
&\Pr[H(G) = N_y \text{ and } N_z(G) = N_z \mid E \text{ and } \mathcal{O}] \\
&= \Pr[H(G) = N_z \text{ and } N_z(G) = N_z \mid E \text{ and } \mathcal{O}] \\
&= \Pr[H(G) = N_z \mid E \text{ and } \mathcal{O}]
\end{aligned}$$

The above ratio then becomes

$$\begin{aligned}
&\frac{\Pr\left[\vec{\ell}(G) = \vec{\ell}_z \mid E \text{ and } \mathcal{O}\right]}{\Pr\left[\vec{\ell}(G) = \vec{\ell}_y \mid E \text{ and } \mathcal{O}\right]} \\
&= \frac{\sum_{N_z \notin \mathcal{A}_z} \Pr[H(G) = N_z \mid E \text{ and } \mathcal{O}]}{\Pr[H(G) = N_y \mid E \text{ and } \mathcal{O}]} \\
&= \frac{\sum_{N_z \notin \mathcal{A}_z} \Pr[H(G) = N_y \text{ and } N_z(G) = N_z \mid E \text{ and } \mathcal{O}]}{\Pr[H(G) = N_y \mid E \text{ and } \mathcal{O}]} \\
&= \Pr[N_z(G) \notin \mathcal{A}_z \mid H(G) = N_y \text{ and } E \text{ and } \mathcal{O}]
\end{aligned}$$

The input distribution $\mathcal{B}_k$ first chooses the hard paths. Then every other edges is added independently at random. If $N_z(G)$ is not a hard path, at each level $i \in [2..D]$, its node is chosen from the $2^k\chi/2$ easy nodes at this level. A sufficient condition for $N_z(G)$ not to be in $\mathcal{A}_z$ is that for all its edges not fixed by E, their destinations do not collide with any node mentioned in E. Let h_i be the number of nodes mentioned in E at level i that $N_z(G)$ must avoid. It follows that

$$\begin{aligned}
&\Pr[N_z(G) \notin \mathcal{A}_z \mid H(G) = N_y \text{ and } E \text{ and } \mathcal{O}] \\
&\geq \Pi_{i \in [2..D]}(1 - \frac{h_i}{2^k\chi/2}) \\
&\geq 1 - \frac{\sum_{i \in [2..D]} h_i}{2^k\chi/2} \;\geq\; 1 - \frac{2 \cdot \chi/8}{2^k\chi/2} \;\geq\; \frac{3}{4}
\end{aligned}$$

because $\sum_{i\in[2..D]} h_i \leq \chi/8$ as E contains at most $\chi/8$ different edges and each edge involves 2 nodes. ∎

Claim 4 *For any computation path γ in A,*

$$Pr_{G\in\mathcal{B}_k}\left[\, w''_\gamma(G) \geq S/\log n \mid G \in \mathcal{B}_k(E_\gamma)\,\right] \leq 2^{-9S}.$$

Proof: Recall that $w''_\gamma(G)$ is the number of roots r in G such that r is a low collision root with respect to γ and $Prog_{\langle\gamma,r\rangle}(G)$ is true. Hence, the expected value μ of $w''_\gamma(G)$ is

$$\sum_{\text{low collision roots r}} \Pr_{G\in\mathcal{B}_k}\left[\, Prog_{\langle\gamma,r\rangle}(G) \mid G \in \mathcal{B}_k(E_\gamma)\,\right]$$

and by Claim 3,

$$\begin{aligned}
\mu &\leq \sum_{\text{low collision root r}} \frac{4}{3} \times \frac{y_{\langle\gamma,r\rangle}}{y_{\langle\gamma,r\rangle} + z_{\langle\gamma,r\rangle}} \\
&\leq \frac{4}{3}\frac{\sum_{\text{low collision root r}} y_{\langle\gamma,r\rangle}}{f^{D'}} \\
&\leq \frac{\chi}{6\cdot f^{D'}},
\end{aligned}$$

as $\sum_r y_{\langle\gamma,r\rangle} \leq \chi/8$ (at most $\chi/8$ different edges are queried by γ.) Since $\chi \in O(n)$ and $f^{D'} \geq n^{10}$, we have $\mu \in O(n^{-9})$. By Lemma 12.1,

$$\begin{aligned}
&\Pr_{G\in\mathcal{B}_k}\left[\, w''_\gamma(G) \geq \frac{S}{\log n} \mid G \in \mathcal{B}_k(E_\gamma)\,\right] \\
&\leq 2^{-(\frac{S}{\log n})(\log(\frac{S}{\mu\log n})-\log e)} \\
&\leq 2^{-9S}.
\end{aligned}$$

∎

13 Conclusion

We have proven that any 2-sided probabilistic NNJAG solving the st-connectivity problem for n-node graphs in (expected) time T using space S must have $T \in 2^{\Omega(\log^2(n/S))}$ when $S \in O(n^{1-\delta})$, for some $\delta > 0$, and $T \in 2^{\Omega(\log^2(\frac{n\log n}{S})/\log\log n)} \times (nS/\log n)^{1/2}$ for general $S \in O(n\log n)$. This greatly improves the previous bounds of $ST \in \Omega(n^2/\log n)$ by Barnes and Edmonds [BE93] and $S^{1/3}T \in \Omega(n^{4/3})$ by Edmonds [Edm93a]. Moreover, the bound is tight for $S \in n^{1-\Omega(1)}$. As a corollary, we also obtained a space lower bound of $\Omega(\log^2 n)$ on a probabilistic NNJAG. No such tight lower bound was known before even in the more restricted JAG model.

An obvious open problem is to close the gap between the upper and lower bounds when $S \notin n^{1-\Omega(1)}$. However, the major open problem is to prove similar lower bounds on a general model of computation. To acheive that, one possible approach is to start with a JAG/NNJAG-like model and add more and more power, pushing our way towards the ultimate model of the branching program. A major complaint to a JAG or NNJAG is its restricted access to the inputs. As pointed out in Etessami and Immerman [EI94], the space lower bounds of [CR80, BS83, Poo93] are proven on a tree. However, it is easy for a RAM to solve STCON on trees in $O(\log n)$ space. All it needs to do is to walk a "pebble" from node t backward and see if it hits node s.

In response to this, we define a model called the *Stack NNJAG* that can solve STCON for trees in $O(\log n)$ space and yet on this model we can still prove the same time-space lower bound. In this model, there is a constant number of *stack* pebbles in addition to those regular pebbles. Each stack pebble has a stack which can remember the path that it has traversed since its last jump. More precisely, all the pebbles, whether the regular or the stack ones, are initially on node s. The stack of each stack pebble is empty initially. Whenever a stack pebble walks along an edge (u, v), the node u is pushed onto the stack. Whenever a stack pebble jumps to another pebble P', it empties its stack. If P' is also a stack pebble, then P copies the stack of P' to its own stack. A stack pebble can also backtrack along the path, i.e., to move to the node v if v is the top of the stack and then pop the stack. Note that the pebble is not allowed to visit any arbitrary node. Any node reachable by a stack pebble must be reachable from s by a directed path. The space for storing the stacks is given for free.

To prove the time-space lower bound, observe that the height of the graph used in our paper is $O(\sqrt{(n \log n)/S})$. If $\sqrt{(n \log n)/S} \leq S/\log n$, each stack can only store at most $O(S/\log n)$ nodes. Since a Stack NNJAG has a constant number of Stack pebbles, it can be simulated by a normal NNJAG with at most $\Theta(S/\log n)$ extra pebbles. The extra pebbles simply jump to and remain on each node that a stack pebble reaches. This increases the space used by the algorithm by at most $\Theta(S)$. If $\sqrt{(n \log n)/S} \geq S/\log n$, then $S \leq n^{1/3} \log n$. In this case, the bound we have for a normal NNJAG is $T = 2^{\Omega(\log^2 n)}$. Now observe that the height of each stack is at most the height of the graph, i.e., at most $O(\sqrt{n \log n})$. Hence any Stack NNJAG with space S can be simulated by a normal NNJAG with space $O(S + \sqrt{n \log n}) \in O(\sqrt{n \log n})$. Hence the same lower bound applies.

Note that the Stack NNJAG model seems to be incomparable with a branching program because of the way we charge the space. Also, defining an intermediate model between the NNJAG model and branching program seems hard. For example, allowing the model to move a pebble to an arbitrary node or to the next node in some fixed ordering would give the power of branching programs. Within a polynomial factor of time and constant factor of space, so does allowing it to move pebble backwards along any directed edge [BBR+90]. The idea is that one can treat the graph as undirected and using a universal traversal sequence [AKL+79], visit any vertex in polynomial

time. Hence whenever the branching program queries the out-edges of a node v, the enchanced NNJAG can place a pebble on node v (by the universal traversal sequence) and perform the same query on v.

Acknowledgments

We are especially grateful to Greg Barnes for his invaluable insights. We thank Allan Borodin, Faith Fich, Charles Rackoff and Hisao Tamaki for their helpful discussions and support. We also thank the annonymous referees for their careful reading and helpful comments. Last but not the least, we thank Johan Hastad for pointing out a bug in the original proof of Lemma 12.1.

References

[AKL+79] 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.

[AN72] K. B. Athreya and P. E. Ney, editors. *Branching Processes*. Springer-verlag, Berlin, 1972.

[BBR+90] P. Beame, A. Borodin, P. Raghavan, W. L. Ruzzo, and M. Tompa. Time-space tradeoffs for undirected graph connectivity. In *31st Annual Symposium on Foundations of Computer Science*, pages 429–438, St. Louis, MO, October 1990. IEEE. Full version submitted for journal publication.

[BBRS92] Greg Barnes, Jonathan F. Buss, Walter L. Ruzzo, and Baruch Schieber. A sublinear space, polynomial time algorithm for directed s-t connectivity. In *Proceedings, Structure in Complexity Theory, Seventh Annual Conference*, pages 27–33, Boston, MA, June 1992. IEEE.

[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.

[BE93] Greg Barnes and Jeff Edmonds. Time-space lower bounds for directed s-t connectivity on JAG models. In *34th Annual Symposium on Foundations of Computer Science*, pages 228–237, Palo Alto, CA, November 1993.

[Bea91] P. Beame. A general time-space tradeoff for finding unique elements. *SIAM Journal on Computing*, 20(2):270–277, 1991.

[BFK+81] 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.

[BFMadH+87] 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.

[Bor82] Allan Borodin. Structured vs. general models in computational complexity. *L'Enseignement Mathématique*, XXVIII(3-4):171–190, July-December 1982. Also in [L'E82, pages 47–65].

[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.

[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.

[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.

[Edm93a] Jeff Edmonds. *Time-Space Lower Bounds for Undirected and Directed ST-Connectivity on JAG Models.* PhD thesis, University of Toronto, 1993.

[Edm93b] Jeff Edmonds. Time-space trade-offs for undirected *st*-connectivity on a JAG. In *Proceedings of the Twenty Fifth Annual ACM Symposium on Theory of Computing*, pages 718–727, San Diego, CA, May 1993.

[EI94] Kousha Etessami and Neil Immerman. Reachability and the power of local ordering. In *11th Annual Symposium on Theoretical Aspects of Computer Science*, pages 123–135, February 1994. Springer-Verlag LNCS 775.

[EP95] Jeff Edmonds and Chung Keung Poon. A nearly optimal time-space lower bound for directed *st*-connectivity on the NNJAG model. In *Proceedings of the Twenty Seventh Annual ACM Symposium on Theory of Computing*, pages 147–156, Las Vegas, NV, May 1995.

[Gil77] J. Gill. Computational complexity of probabilistic Turing machines. *SIAM Journal on Computing*, 6(4):675–695, December 1977.

[Imm88] Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, October 1988.

[Joh90] David S. Johnson. A catalog of complexity classes. In Jan van Leeuwan, editor, *Handbook of Theoretical Computer Science*, volume A: Algorithms and Complexity, chapter 2, pages 67–161. M.I.T. Press/Elsevier, 1990.

[L'E82] *Logic and Algorithmic,* An International Symposium Held in Honor of Ernst Specker, Zürich, February 5–11, 1980. Monographie No. 30 de L'Enseignement Mathématique, Université de Genève, 1982.

[LP82] H. R. Lewis and C. H. Papadimitriou. Symmetric space-bounded computation. *Theoretical Computer Science*, 19(2):161–187, August 1982.

[Nis92] 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.

[Pip92] Nicholas Pippenger. The asymptotic optimality of spider-web networks. *Discrete Applied Mathematics*, 37/38:437–450, 1992.

[Poo93] Chung Keung Poon. Space bounds for graph connectivity problems on node-named JAGs and node-ordered JAGs. In *34th Annual Symposium on Foundations of Computer Science*, pages 218–227, Palo Alto, CA, November 1993.

[Poo96] Chung Keung Poon. *On the Complexity of the ST-Connectivity Problem.* PhD thesis, University of Toronto, 1996.

[Sav70] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of Computer and System Sciences*, 4(2):177–192, 1970.

[Sze88] Róbert Szelepcsényi. The method of forcing for nondeterministic automata. *Acta Informatica*, 26:279–284, 1988.

[Tom82] Martin Tompa. Two familiar transitive closure algorithms which admit no polynomial time, sublinear space implementations. *SIAM Journal on Computing*, 11(1):130–137, February 1982.

[Wig92] Avi Wigderson. The complexity of graph connectivity. In I. M. Havel and V. Koubek, editors, *17th Symp. Mathematical Foundations of Computer Science*, pages 112–132. Springer-Verlag LNCS 629, August 1992.

[Yao77] 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.

[Yao88] 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.