**CS 860: Space Bounded Computation** **University of Waterloo: Winter 2026**

# Lecture 5: (Node-Named) Jumping Automata on Graphs

February 10, 2026

*Instructor: Sepehr Assadi* *Scribe: Gunadi Gani*

## Topics of this Lecture

**1 NNJAG: Node Named Jumping Automaton on Graphs** 1

**2 A Hard Input Construction: Skinny Trees** 2

**3 Warmup: When Pebbles Do Not Collide** 6

**4 The Main Inductive Argument** 11

In this lecture, we cover a result of [Poo93] establishing a near-optimality of Savitch's Theorem for STCONN from the last lecture, in a restricted model of computation called NNJAG.

## 1 NNJAG: Node Named Jumping Automaton on Graphs

Recall the STCONN problem from the previous lecture: given a directed graph $G = (V, E)$ and vertices $s, t \in V$, can $s$ reach $t$ or not. We consider the following restricted model of computation for this problem.

**Definition 1.** The **Node-Named Jumping Automaton on Graphs (NNJAG)** model is defined as follows.

We have $p$ **pebbles** $P_1, \cdots, P_p$ and $q$ **states** $Q_1, \cdots, Q_q$ and in input $n$-vertex graph $G = (V, E)$ where $V := \{1, \ldots, n\}$, where, for simplicity, we assume $s = 1$ and $t = n$. A **configuration** consist of a pair $(Q, \Pi)$ where $Q$ is one of the $q$ possible states and $\Pi : [p] \to [n]$ maps each pebble to one of the vertices. We also assume that outgoing edges of each vertex in $G$ is labelled uniquely.

The original configuration is $(Q_1, \Pi_{start})$ where $\Pi_{start}$ maps all pebbles to the vertex $s$. The automaton moves through the configurations according to two types of **transitions**: a pebble **move** in the direction of an edge of the graph, and a pebble jump to the location of another pebble (we formalize this further below). The goal is to reach a configuration wherein at least one pebble is on $t$ if $s$ can reach $t$ in $G$, or the state is $Q_q$ (a "reject" state) if $s$ cannot reach $t$ in $G$.

Finally, we consider the **space** of an automaton as $p \log n + \log q$ (we will elaborate on this later).

**Transitions.** There are two possible *transitions* in the NNJAG model:

1. A **move** is defined by a pebble $i \in [p]$, a number $j \in [n]$ and a new state $Q' \in \{Q_1, \cdots, Q_q\}$ which updates the configuration $(Q, \Pi) \to (Q', \Pi')$ as follows:

- Set $\Pi'(i) = v$ where $v$ is the vertex at the end of the outgoing edge $\Pi(i)$ with label $j$ (effectively, we move pebble $P_i$ to vertex $v$); if vertex $\Pi(i)$ does not have a $j$-edge, we consider $v = \Pi(i)$ (i.e., the vertex remains).
- Set $\Pi'(k) = \Pi(k)$ for all $k \neq i$ (we keep all other pebbles at the same vertex).

2. A **jump** is defined by two pebbles $i, j \in [p]$ and a new state $Q' \in \{Q_1, \cdots, Q_q\}$ which updates the configuration $(Q, \Pi) \to (Q', \Pi')$ as follows:
   - Set $\Pi'(i) = \Pi(j)$ (we move pebble $P_i$ to the vertex in which pebble $P_j$ is located).
   - Set $\Pi'(k) = \Pi(k)$ for all $k \neq i$ (we keep all other pebbles the same).

**Space.** As we mentioned earlier, we consider the space of a NNJAG $J$ as $p \log n + \log q$. This is because we can specify a fixed configuration in $O(p \log n + \log q)$ bits of space. Note that in general, NNJAGs form a non-uniform model of computation – to specify an NNJAG, we simply write an instruction-set that for every choice of a configuration $(Q, \Pi)$, specifies what next transition is; namely, the instruction-set has size $q \cdot n^p$ (and thus, the "algorithm description" itself depends on $n$).

The non-uniformity of NNJAGs means there is no hope of simulating an NNJAG with a uniform algorithm. However, it is not hard to verify that we can indeed simulate an NNJAG of space $p \log n + \log q$ with a *branching program* of $O(p \log n + \log q)$ space. On the other hand, one can also use NNJAGs to simulate algorithms such as DFS/BFS as well as Savitch's theorem in a similar space (namely, $O(n \log n)$ in the case of the former two and $O(\log^2 n)$ in the case of the latter). We will not prove this formally in this course but the general idea is to use the pebbles to keep track of a stack (in case of DFS and Savitch's algorithm) or queue (in case of BFS) to be able to navigate the graph (the states allow one to keep track of the variables and information in the algorithm).

**Main result.** The main result we are interested in this lecture is the following lower bound of [Poo93].

**Theorem 2** ([Poo93])**.** *Any* NNJAG *for* STCONN *on $n$-vertex graphs requires $\Omega(\frac{\log^2 n}{\log \log n})$ space.*

This lower bound has been improved to the *optimal* $\Omega(\log^2 n)$ space subsequently in [EPA99] (who also proved the (near) optimality of the BBRS algorithm we saw in Lecture 4 in the NNJAG model).

## 2 A Hard Input Construction: Skinny Trees

We now define a family of hard instances. For integers $p, q \geqslant 1$ (corresponding to pebbles and states in the NNJAG), the family of **skinny trees** is defined recursively. Firstly, define the parameter

$$d = 18p^2 \log(pq) \qquad \text{(so } d \approx p^2 \log q\text{)}. \tag{1}$$

**Skinny trees (base case).** We denote $G(x)$ for $x \in \{0,1\}^d$ to be the graph with $2d+1$ vertices

$$\{s\} \sqcup \{v_1^0, \ldots, v_d^0\} \sqcup \{v_1^1, \ldots, v_1^1\}.$$

The directed edges are from:

$$s \to v_1^0 \;,\; s \to v_1^1 \quad \text{, and} \qquad v_i^{x_i} \to v_{i+1}^0 \;,\; v_i^{x_i} \to v_{i+1}^1 \text{ for all } i \in [d-1].$$

(In terms of labelings, any edge going to some $v_*^0$ (resp. $v_*^1$) is labeled 0 (resp. 1)).

The vertex $s$ is called the **start** of $G(x)$ and the vertex $v_d^{x_d}$ is called the **target** of the graph $G(x)$.

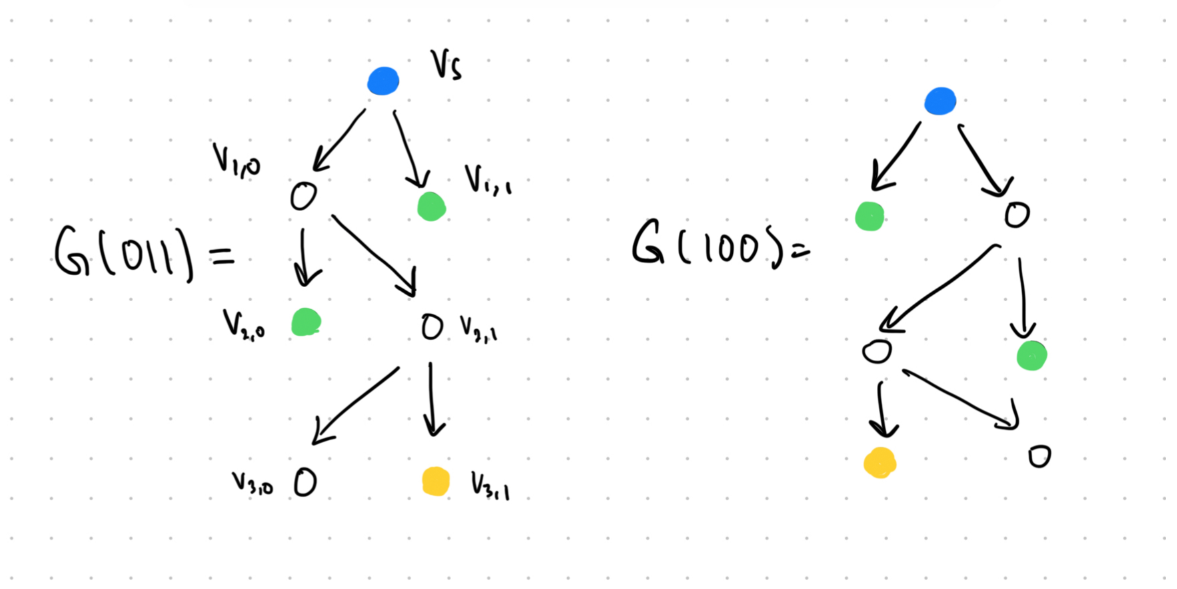

Figure 1: Example of $G(011)$ and $G(100)$ where the blue vertex is the start and yellow is the target and green is a leaf, namely, a vertex with no outgoing edges.

See Figure 5 for an illustration of the base case skinny trees.

We can now specify the recursive construction.

**Skinny trees (inductive case).** For any integer $k \geqslant 2$, we denote $G(x_1, \dots, x_k)$ for $x_1, \dots, x_k \in \{0,1\}^d$ to be the following graph.

1. Create a $G(x_k)$ as in the base case.
2. Replace each vertex of $G(x_k)$ with a copy of $G(x_1, x_2, \cdots, x_{k-1})$.
3. The copy of $G(x_1, \cdots, x_{k-1})$ which replaces a vertex $v$ of $G(x_k)$ is denoted by $G^v$ of $G(x_1, \cdots, x_k)$.
4. For any edge $u$ to $v$ in $G(x_k)$, we connect the target vertex of $G^u$ to the start vertex of $G^v$.
5. Define $G^{start}$ of $G(x_1, \cdots, x_k)$ as the copy of $G(x_1, \cdots, x_{k-1})$ that replaces $s$ of $G(x_k)$; $G^{target}$ of $G(x_1, \cdots, x_{k-1})$ is the copy of $G(x_1, \cdots, x_k)$ which replaces the target of $G(x_k)$. The start of $G(x_1, \cdots, x_k)$ is the start of $G^{start}$ and the target of $G(x_1, \cdots, x_k)$ is the target of $G^{target}$.

As a matter of notation, we refer to a vertex of a skinny tree as a **leaf** if there are no outgoing edges from $v$. Similarly, we refer to a copy of $G^v$ in a skinny tree as a **leaf-copy** if its target is a leaf.

See Figure 2 and Figure 3 for further illustrations.

Skinny trees will be used to prove the following theorem, which formalizes Theorem 2.

**Theorem 3** ([Poo93]). *For all NNJAG $J$ with $p$ pebbles and $q$ states, there exist a graph $G$ with at most $(36p^2 \log(pq) + 1)^p$ nodes and two special nodes $s$ and $t$ in $G$ such that if $J$ starts with all pebbles on node $s$, no pebble will ever visit node $t$ during the computation.*

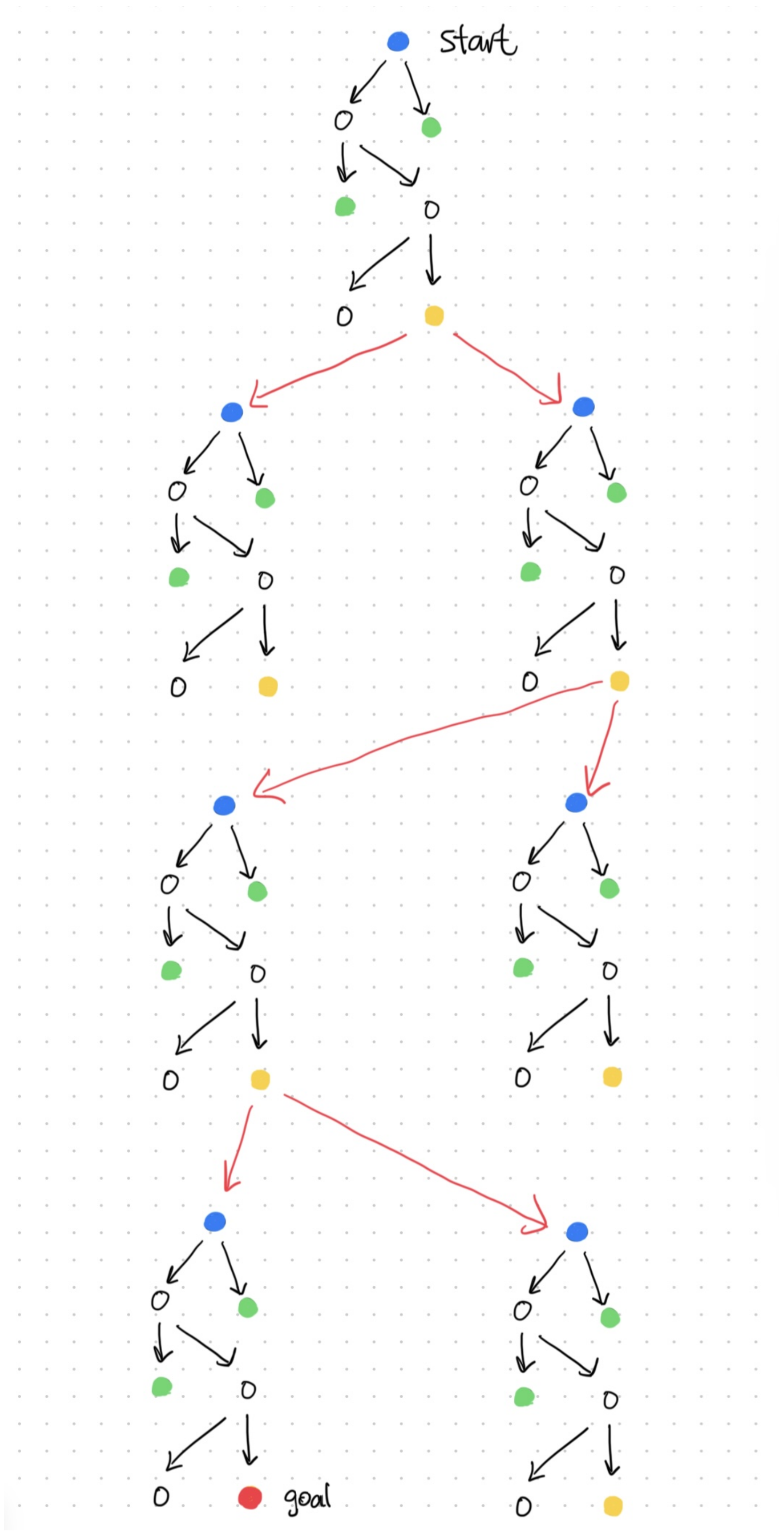

Figure 2: Example of $G(011, 100)$ where the red edges denote the edges of $G(100)$ and the black edges denote the edges of the copies of $G(011)$ due to the recursive definition of $G(011, 100)$

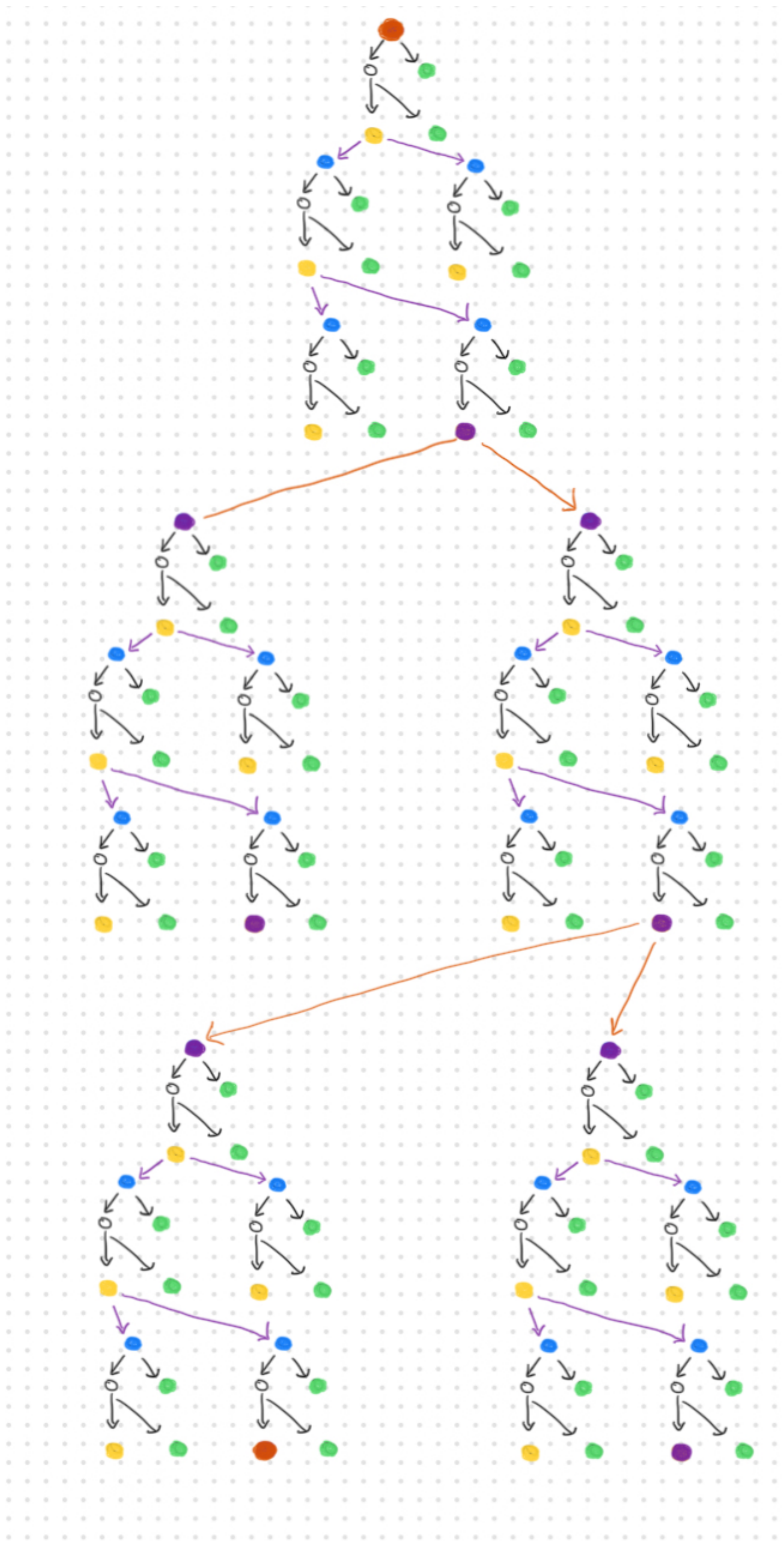
Figure 3: Example of $G(00, 01, 10)$.

Before moving on, we encourage the reader to consider the following two problems: (1) can an NNJAG navigate all graphs $G(x)$ for $x \in \{0,1\}^d$ with a *single* pebble (regardless of the number of states)? What about two pebbles? (2) can an NNJAG navigate all graphs $G(x_1, x_2)$ for $x_1, x_2 \in \{0,1\}^d$ with two pebbles and arbitrary number of states? what about with $o(\log d)$ many states? What about three pebbles?

Let us now show how Theorem 3 implies Theorem 2 immediately (the proof is just by plugging in the parameters).

*Proof of Theorem 2 from Theorem 3.* Suppose we have an NNJAG $J$ for STCONN with $p$ pebbles and $q$ states and space $o(\frac{\log^2 n}{\log\log n})$. This in particular implies that $p < \log n$ and $q < n^{\log n}$. By Theorem 3, $J$ cannot solve STCONN on skinny trees with parameters $p$ and $q$ which have $n = (36p^2 \log(pq) + 1)^p$ vertices. We now only need to bound $p$ in terms of $n$ to conclude the proof. We have,

$$\begin{aligned}
\log n &< p \log(36p^2 \log(pq) + 1) && \text{(taking log on both sides of the equation for } n\text{)}\\
&< p \cdot (\log 37 + 2\log p + \log\log pq) && \text{(as } 1 \leqslant p^2 \log(pq) \text{ and } \log(ab) = \log a + \log b\text{)}\\
&< p \cdot (6 + 2\log p + \log\log p + \log\log q) \\
& && \text{(as } \log\log pq = \log(\log p + \log q) \leqslant \log\log p + \log\log q \text{ and } \log 37 < 6\text{)}\\
&< p \cdot (6 \log\log n) && \text{(as } p < \log n, \log q < \log^2 n\text{)}
\end{aligned}$$

which implies

$$p = \Omega(\frac{\log n}{6\log\log n}).$$

But then this implies that the space of $J$ needs to be $\Omega(\frac{\log^2 n}{\log\log n})$, a contradiction, and concluding the proof. □

## 3 Warmup: When Pebbles Do Not Collide

We start the proof of Theorem 3 now by consider a simple especial case as a warm-up: when the entire computation does not lead to any "collision" between the pebbles (we will define this formally soon). This warm-up also will work as the induction base and proof strategy of the main proof and captures almost all the main ideas.

Throughout, fix an NNJAG $J$ with $p$ pebbles and $q$ states.

**Definition 4.** Let $Q$ be any state, $\Pi$ denote the mapping of pebbles to vertices and $x_1, x_2, \cdots, x_p$ be in $\{0,1\}^d$ each. We define a **1-computation** denoted by

$$C_1(Q, \Pi, x_1, x_2, \cdots, x_p)$$

to be the shortest sequence of transitions (of moves and jumps) obtained by running $J$ on $G(x_1, x_2, \cdots, x_p)$ starting from $(Q, \Pi)$ until some copy of $G(x_1)$ is **traversed** or two of the pebbles collide, i.e., reside on the same vertex (if $\Pi$ has colliding pebbles, the 1-computation terminate right at the start). A copy of $G(x_1)$ (denoted by $G^u$) is traversed in a 1-computation if there is a pebble which initially placed either at the start of $G^u$ or at some vertex above it and reaches the goal of $G^u$ in the 1-computation.

The lemma we would like to prove in this section is the following (recall that $J$ is fixed and we do not repeat this in each lemma).

**Lemma 5.** *There exists a choice of $y \in \{0,1\}^d$ such that for all $(Q, \Pi)$ and all $x_2, \ldots, x_p \in \{0,1\}^d$, the 1-computation $C_1(Q, \Pi, y, x_2, \ldots, x_p)$ is not traversing a copy of $G(y)$ (thus ends due to a pebble collision).*

We start by showing that 1-computations can be characterized using much less information than the entire tuple $(Q, \Pi, x_1, \ldots, x_p)$. To do this, we need the following definition.

**Definition 6.** For a fixed $(Q, \Pi)$ and $y, x_2, \ldots, x_p \in \{0,1\}^d$, define $L := L(x_2, \cdots, x_p)$ as a string in $\{0,1\}^p$ such that $L[i] = 1$ iff the copy of $G(y)$ in $G(y, x_2, \ldots, x_p)$ that the pebble $P_i$ lies in (based on $\Pi$) is a leaf copy, namely, there are no edges leaving this copy.

Notice that, by construction of skinny trees, the choice of $L$ only depends on $x_2, \ldots, x_p$ and not $y$ itself. Thus, writing $L = L(x_2, \ldots, x_p)$ is indeed well-defined above. To give a example of $L$, consider $G(00, 01, 10)$ (as in Figure 3) and $G(00, 01, 00)$ and place pebble $P_1$ in the middle right copy of $G(00, 01)$ in both graphs in the goal (colored purple) of that copy. In $G(00, 01, 10)$ this pebble is not in a leaf-copy (hence $L[1] = 0$) and in $G(00, 01, 00)$ this pebble is in a leaf-copy (hence $L[1] = 1$).

The following observation allows us to characterize 1-computations more succinctly.

**Observation 7.** *For any $y \in \{0,1\}^d$, and $x_2, x_3, \cdots, x_p \in \{0,1\}^d$ and $x'_2, x'_3, \cdots, x'_p \in \{0,1\}^d$, if*

$$L(x_2, x_3, \cdots, x_p) = L(x'_2, x'_3, \cdots, x'_p)$$

*then*

$$C_1(Q, \Pi, y, x_2, x_3, \cdots, x_p) = C_1(Q, \Pi, y, x'_2, x'_3, \cdots, x'_p).$$

*Proof.* We claim that *as long as* no copy of $G(y)$ in either graphs $G(y, x_2, \ldots, x_p)$ or $G(y, x'_2, \ldots, x'_p)$ is traversed, then:

1. Moving a pebble $P$ along a $j$-edge places $P_i$ on the same vertex in both graphs.
2. Jumping a pebble $P_i$ into pebble $P_j$ places $P_i$ on the same vertex in both graphs.

By applying the claim to each operation of $J$, we can conclude the proof of the observation.

*Proof of part 1.* During a 1-computation each pebble $P_i$ is either in the copy $G^u$ of $G(y)$ it started in $\Pi$ or in the copy $G^v$ of $G(y)$ one level below it (if it jumps the 1-computation finishes because two pebbles collide, and if it goes to an even lower copy, then it has traversed a $G(y)$ and the 1-computation finishes).

In the case where $P_i$ is completely inside $G^u$ or $G^v$ (i.e., is not on their targets), then the claim immediately follows since the innermost graph is the same in both $G(y, x_2, \cdots, x_p)$ and $G(y, x'_2, \cdots, x'_p)$. If $P_i$ is at the goal of $G^u$, then having $L(x_2, \ldots, x_p) = L(x'_2, \ldots, x'_p)$, again means the next move of $P_i$ will be the same (note that in skinny trees, a vertex being a leaf or not uniquely determines the edges going out of it). Finally, if $P_i$ is at the goal of $G^v$, a traverse has happened and the 1-computation finishes now.

*Proof of part 2.* Since the pebbles start at the same vertices in both graphs (according to $\Pi$) and move the same (by part 1), and a jump moves a pebble to an already pebbled vertex, jumps cannot change based on the underlying graph. This concludes the proof of the observation. □

By this lemma, we may extend the definition of 1-computation to only be defined by $Q, \Pi, y \in \{0,1\}^d$ and $L \in \{0,1\}^p$ where

$$C_1(Q, \Pi, y, L) := C_1(Q, \Pi, y, x_2, \cdots, x_p)$$

for any (hence all due to Observation 7) $x_2, \cdots, x_p \in \{0,1\}^d$ such that $L(x_2, \cdots, x_p) = L$.

**Proof idea of Lemma 5.** We can now sketch the proof idea. Recall that our goal in Lemma 5 (using the characterization above) is to find a choice of $y$ such that no matter the choice of $Q, \Pi, L$, $C_1(Q, \Pi, y, L)$ is not traversing a copy of $G(y)$. We show this by bounding the number of 1-computations that do traverse and show that this is smaller than $2^d$, which means that there are some $y$'s that no matter the choice of $Q, \Pi, L$, will not create a 1-computation that can traverse. The general idea here is that in order for $J$ to traverse a copy of $G(y)$, it needs to "explore" almost $d$ different vertices; but, moving a pebble to a vertex comes with a risk that this vertex might be a leaf-node. Since in a 1-computation a jump cannot happen, moving a pebble to a leaf-node "eliminates" this pebble. So for any fixed choices of $Q, \Pi, L$, we will be able to bound the number of $y$'s where $C_1(Q, \Pi, y, L)$ can traverse. Since the total number of choices of $(Q, \Pi, L)$ is upper bounded by $q \cdot n^p \cdot 2^p$ which is $\ll 2^d$ by our choice of $d$, we will be able to conclude the proof.

**Formalizing the proof of Lemma 5.** To continue, we need some more definition.

**Definition 8.** Fix $(Q, \Pi, L)$ and $y \in \{0,1\}^d$. Let $i \in [d]$ be the index of some bit in $y$. We say that:

1. bit $i$ is **known** iff there exist a pebble $P$ on $v_i^0$ or $v_i^1$ of some copy of $G(y)$ in $\Pi$.
2. bit $i$ is **traversed** iff it is not known and there exist some pebble $P$ that visited $v_i^0$ or $v_i^1$ of some copy of $G(y)$ during $C_1(Q, \Pi, y, L)$.
3. bit $i$ is **free** iff it is neither known nor traversed.

Finally, for a 1-computation $C_1(Q, \Pi, y, L)$ and a time $t$ during the computation, we define $KT_t(Q, \Pi, y, L)$ as the set of bits that are known or traversed by the time $t$.

Note that at most $p$ bits can be known in a 1-computation. On the other hand, for a 1-computation to traverse a copy of $G(y)$, the NNJAG $J$ needs to traverse all remaining $\geqslant d - p$ bits.

**Definition 9.** A pebble $P$ is an **active** pebble if it is not sitting at leaf vertex; otherwise, it is **inactive**.

Note that we can start with at most $p$ active pebbles and each move that traverse a (new) bit has "equal chance" of making an active pebble inactive instead. Lemma 10 below formalizes this.

Define the function $f(d', p')$ for all integers $d', p' \geqslant 0$ as

$$f(d', p') = \begin{cases} f(d'-1, p') + f(d'-1, p'-1) & \text{when } d', p' > 0 \\ 1 & \text{when } d' = 0, p' > 0 \\ 0 & \text{when } d' \geqslant 0, p' = 0 \end{cases}.$$

See Table 1 for some initial values of $f$.

| $d' \setminus p'$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 2 | 2 | 2 |
| 2 | 0 | 1 | 3 | 4 | 4 |
| 3 | 0 | 1 | 4 | 7 | 8 |
| 4 | 0 | 1 | 5 | 11 | 15 |

Table 1: Entries of $f(d', p')$ for $d', p' \in [4]$.

We can see by induction that $f(d, p) \leqslant d^p$.

Let $S \subseteq \{0,1\}^d$ and $B \subseteq [d]$. We say $S$ is **fixed** on $B$ iff

$$\text{for all } y, z \in S\ ,\ y_B = z_B \qquad (\text{meaning } y_i = z_i\ \forall i \in B).$$

The following lemma says that for a fixed $(Q, \Pi, L)$ and some time $t \geqslant 1$, if we have a set $S \subseteq \{0,1\}^d$ as potential choices for $y$ such that all of them have the same known and traverse bits at this time (same $KT_t$), both in location and value of the bit, and all of them have $d'$ free bits and most $p'$ active pebbles, then, the number of $y$ in $S$ that $J$ can traverse is at most $f(d', p')$.

**Lemma 10** (Counting Lemma)**.** *Let $(Q, \Pi, L)$ be arbitrary, $d' \in \{0, 1, \cdots, d\}$ and $p' \in \{0, 1, \cdots, p\}$, and $t \geqslant 1$. Fix a set $S \subseteq \{0, 1\}^d$ and suppose:*

1. *for all $y \in S$, $KT_t(Q, \Pi, y, L)$ is the same set $B \subseteq [d]$;*
2. *$S$ is fixed on $B$; and,*
3. *$C_1(Q, \Pi, y, L)$ has $d'$ free bits and at most $p'$ active pebbles for all $y \in S$.*

*Then,*

$$\#\text{ of } y \in S \text{ such that } C_1(Q, \Pi, y, L) \text{ traverses} \leqslant f(d', p').$$

*Proof.* Note that fixing $KT_t = B$ and having $S$ fixed on $B$ implies that the structure of innermost graph (namely, copies of $G(y)$) at the vertices $v_i^0, v_i^1$ where bit $i$ is known or traversed are exactly the same for every $y \in S$ (see Figure 4 for an illustration). Moreover, define

$$T := \{y \in S \mid C_1(Q, \Pi, y, L) \text{ traverses}\}.$$

The goal is to prove $|T| \leqslant f(d', p')$.

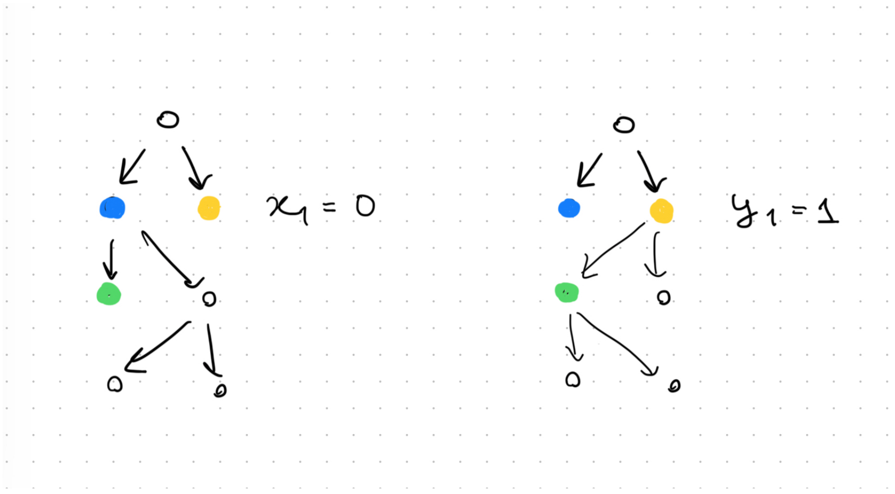

Figure 4: $x$ and $y$ cannot be in the same $S$ in Lemma 10 since bit 1 is known but $x_1 = 0, y_1 = 1$ which means $S$ is not fixed on the known bits.

We prove this by induction on $d'$. The base case of this induction is when either $d' = 0$ or $p' = 0$. When $p' = 0$, then for any $d' \geqslant 0$ there is no $y \in S$ for which $C_1(Q, \Pi, y, L)$ is traversing since there is no active pebble; this in turn means no inactive pebble can be made active anymore (as it creates a collision and finishes the 1-computation). So $|T| = 0 \leqslant f(d', p')$. When $d' = 0$ and $p' > 0$, $|T| \leqslant 1 = f(d', p')$ because fixing $S$ on all not-free bits, which are all of bits here, results in $|S| = 1$.

We assume that the result hold for any $d'', p''$ where $d'' = 0$ or $p'' = 0$ or $d'' < d'$ and $p'' \leqslant p'$. Now assume that $d' > 0, p' > 0$ and $S \subseteq \{0, 1\}^d$ satisfy the conditions of this lemma.

For each $y \in S$, run the 1-computation $C_1(Q, \Pi, y, L)$. Every step of the computation until the first time $t' > t$ where a free bit turns into traversed is the same for all $y \in S$. The 1-computation is the same because it starts from the same $(Q, \Pi)$ and as argued earlier, the fixing of $S$ ensures the parts of the skinny tree where the computation occurred on both graphs is the same.

Let $i$ be the bit that turned from free to traversed at time $t'$ (which is the same in all $y \in T$ as argued above). Then partition $S$ into $S_0$ and $S_1$ where

$$S_0 = \{y \in S \mid y_i = 0\} \quad \text{and} \quad S_1 = \{y \in S \mid y_i = 1\} \quad (= S \setminus S_0).$$

By symmetry, assume that the pebble which turned the $i$-th bit into traverse ends up at $v_i^0$. The analysis for the other case (where it ends up at $v_{i,1}$) is the same by symmetry. Then at time $t'$:

1. The first condition of the lemma still holds at time $t'$ on both $S_0$ and $S_1$ since $KT_{t'}$ for both of them now also includes $\{i\}$, thus $B \leftarrow B \cup \{i\}$.
2. The second condition holds since every $y \in S_0$ satisfies $y_i = 0$, and every $y \in S_1$ satisfies $y_i = 1$ and thus $S0$ and $S1$ are, individually, fixed on the new set $B$.
3. For all $y \in S_0$, there is $d' - 1$ free bits and $\leqslant p'$ active pebble at time $t'$ of $C_1(Q, \Pi, y, L)$ since bit $i$ is no longer free. For all $y \in S_1$, there is $d' - 1$ free bits and $\leqslant p' - 1$ active pebble at time $t'$ of $C_1(Q, \Pi, y, L)$ since bit $i$ is no longer free and pebble $P$ which flipped the $i$-th bit is no-longer active (since we assumed to be at $v_i^0$ which is a leaf in $G(y)$ for all $y \in S_1$).

Define $T_0 := T \cap S_0$ and $T_1 := T \cap S_1$. We apply the induction step to $d' - 1$ free bits and $\leqslant p'$ active pebbles on $S_0$ which gives $|T_0| \leqslant f(d' - 1, p')$ and to $d' - 1$ free bits and $\leqslant p' - 1$ active pebbles on $T_1$ which gives $|T_1| \leqslant f(d' - 1, p' - 1)$. So $|T| = |T_0| + |T_1| \leqslant f(d' - 1, p') + f(d' - 1, p' - 1) = f(d', p')$ as desired. □

We can now conclude the proof of Lemma 5.

*Proof of Lemma 5.* Define the set

$$T := \left\{(Q, \Pi, y, L) \in [q] \times [n]^p \times \{0,1\}^d \times \{0,1\}^p \mid C_1(Q, \Pi, y, L) \text{ traverses a copy of } G(y)\right\}.$$

Suppose towards a contradiction that the lemma is not true, which implies that $|T| \geqslant 2^d$ because it says for each choice of $y$ in $T$, there exists a choice of $Q, \Pi, L$ that leads to $C_1(Q, \Pi, y, L)$ traversing. We will upper bound $T$ in the following to get to a contradiction.

Let $B$ be the set of known bits in $\Pi$ and $b \in \{0,1\}^p$ be a potential assignment to these bits. Define $S_b \subseteq \{0,1\}^d$ such that $S$ is fixed on $B$ to be $b$. Thus,

$$\{0,1\}^d = \sum_{b \in \{0,1\}^p} S_b.$$

We can apply Lemma 10 to $Q, \Pi, L$ and $S_b$ with at most $d$ free bits and $p$ active pebbles to obtain that

$$T_{Q,\Pi,b,L} := \{y \in S_b \mid C_1(Q, \Pi, y, L) \text{ traverses a copy of } G(y)\}$$

has size at most $f(d, p) \leqslant d^p$.

This implies that

$$|T| = \sum_{Q,\Pi,b,L} |T_{Q,\Pi,b,L}| \leqslant q \cdot n^p \cdot 2^p \cdot 2^p \cdot d^p \leqslant q \cdot (2d+1)^{p^2} \cdot (4d)^p,$$

as $n = (2d+1)^p$.

By our choice of $d = 18p^2 \log(pq)$, one can verify that we get to the contradiction

$$|T| \geqslant 2^d > q \cdot (2d+1)^{p^2} \cdot (4d)^p \geqslant |T|.$$

Thus, our original assumption was wrong and the lemma holds. □

## 4 The Main Inductive Argument

We now provide the main proof. A key new technical definition is that of "colorings" that roughly speaking replace the role of single pebbles in our warm-up proof.

**Definition 11.** Fix a starting position of pebbles $\Pi$. We define the **coloring** of the pebble $P_i$ at time $t$ with respect to $\Pi$, denoted by $\text{color}_t^{\Pi}(i)$ as

$$\text{color}_t^{\Pi}(i) = \begin{cases} \Pi(i) & \text{if } t = 0 \\ \text{color}_{t-1}^{\Pi}(j) & \text{if pebble } i \text{ moved or jumped at time } t \text{ to a vertex where pebble } j \text{ is located} \\ \text{color}_{t-1}^{\Pi}(i) & \text{otherwise} \end{cases} .$$

In other words, starting from a position $\Pi$, originally each pebble is colored with the name of the vertex it starts with, and then whenever a pebble $P$ join another pebble $P'$, it receives the color of $P'$. Note that while pebbles can jump, colors cannot, and that the number of colors is always non-increasing (once a color has no pebble, we consider that color removed).

We now define an analogue of 1-computations for any $k \in [p]$. For this, we first have to specify what "traversing" means.

**Definition 12.** Let $(Q, \Pi)$ be arbitrary and $y_1, y_2, \cdots, y_k \in \{0,1\}^d$ and $x_{k+1}, \ldots, x_p \in \{0,1\}^d$. We say a color $c$ **traverses** a copy of $G(y_1 \cdots, y_k)$ (denoted by $G^v$) in a graph $G(y_1, \ldots, y_k, x_{k+1}, \ldots, x_p)$ during a computation $C = ((Q, \Pi), y_1, \ldots, y_k, x_{k+1}, \ldots, x_p)$ iff the following happens:

1. there is a pebble $i \in [p]$ where $\text{color}_0^{\Pi}(i) = c$ and pebble $P_i$ is located on or above the start of $G^v$ at time 0; and,
2. there is a pebble $j \in [p]$ and a time $t$ during the computation such that $\text{color}_t^{\Pi}(j) = c$ and $P_j$ sits on the goal of $G^v$ at time $t$.

A copy of $G(x_1, \cdots, x_k)$ (denoted by $G^v$) is said to have been **traversed** if there is some color $c$ which traversed $G^v$.

We can now define $k$-computations. As before, we are always fixing an NNJAG $J$ with $p$ pebbles and $q$ states throughout.

**Definition 13.** Let $(Q, \Pi)$ be arbitrary and $x_1, \ldots, x_p \in \{0,1\}^d$. For any $k \in [p]$, a ***k*-computation** $C_k(Q, \Pi, x_1, \cdots, x_p)$ is the shortest sequence of transitions obtained by running $J$ on $G(x_1, x_2, \cdots, x_p)$ starting from $(Q, \Pi)$ until either some copy of $G(x_1, \ldots, x_k)$ is traversed or the number of distinct colors among all pebbles reduces to $p - k$.

It is worth pointing out that 1-computations according to this definition and our original definition are the same as for $k = 1$, two pebble colliding is the same as colors reducing to $p - 1$, and the notion of traversing is the same.

We can now state our main lemma.

**Lemma 14.** *For any choice of $k \in [p]$, there exists a choice of $y_1, \ldots, y_k \in \{0,1\}^d$ such that for all $(Q, \Pi)$ and all $x_{k+1}, \ldots, x_p \in \{0,1\}^d$, the $k$-computation $C_k(Q, \Pi, y_1, \ldots, y_k, x_{k+1}, \ldots, x_p)$ is not traversing a copy of $G(y_1, \ldots, y_k)$ (thus ends due to color reduction).*

Let us first note that the case $k = 1$ in Lemma 14 is the same as our warm-up Lemma 5 and thus we can use it as a base case for proving Lemma 14 inductively. Secondly, notice that for $k = p$, Lemma 14 states that there exists a choice of $y_1, \ldots, y_p$ such that starting from any $(Q, \Pi)$, $J$ cannot traverse $G(y_1, \ldots, y_p)$ in a $p$-computation. Specifically, if we set $Q = Q_1$ and $\Pi = \Pi_0$ (that place all pebbles at $s$), this means that $J$ cannot put any of these pebbles at the target of $G(y_1, \ldots, y_p)$ (as otherwise it amounts to traversing). At the same time, since we always at least one color, the number of distinct colors in a $p$-computation can never reach 0 and thus a $p$-computation can only finish once a traverse has happened. Putting all these together, we obtain that there is a skinny tree $G(y_1, \ldots, y_p)$ such that $J$ cannot place any pebble on its target. This proves Theorem 3.

It thus remains to prove Lemma 14. Suppose the lemma holds for some $k \geqslant 1$ and we prove it for $k+1$ (base case is Lemma 5). Fix a choice of $y_1, \ldots, y_k$ satisfying the lemma guarantee. Our goal is to find a choice of $z \in \{0,1\}^d$ such that $y_1, \ldots, y_k, z$ satisfies the lemma guarantee for $k+1$. For the ease of notation, from now on, we write $y_{\leqslant k} := (y_1, \ldots, y_k)$ and fix it throughout the rest of the proof and do not explicitly condition on it in the definitions. As a matter of notation, we also often denote $x_{\geqslant k+2} := (x_{k+2}, \ldots, x_p)$.

We will now define a function analogous to Definition 6 for the skinny tree $G(y_{\leqslant k}, z)$ instead of a vertex.

**Definition 15.** For a fixed $(Q, \Pi)$ and $z, x_{\geqslant k+2} \in \{0,1\}^d$, define $L_{k+1} := L_{k+1}(x_{k+2}, \ldots, x_p)$ as a string in $\{0,1\}^p$ such that $L[i] = 1$ iff the copy $G^v$ of $G(y_{\leqslant k}, z)$ that the pebble $P_i$ lies on is a leaf-copy in $G(y_{\leqslant k}, z, x_{\geqslant k+2})$, which means the target of $G^v$ has no outgoing edges leaving this copy.

The analogue of Observation 7 also holds for $(k+1)$-computations and the string $L_{k+1}$ defined above. As such, we can write a $(k+1)$-computation instead as

$$C_{k+1}(Q, \Pi, z, L_{k+1}),$$

for $z \in \{0,1\}^d$ and $L_{k+1} \in \{0,1\}^p$ (recall that $y_{\leqslant k}$ is already fixed and we do not need to specify it explicitly).

The general strategy of the proof is quite similar to that of Lemma 5. We start with the following generalizations of our previous definitions. The following generalizes Definition 8.

**Definition 16.** Fix $(Q, \Pi, L)$ and $z \in \{0,1\}^d$. Let $i \in [d]$ be the index of some bit in $z$. We say that:

1. bit $i$ is **known** iff there exist a pebble $P$ on $G^{v_i^0}$ or $G^{v_i^1}$ of some copy of $G(y_{\leqslant k}, z)$ at the beginning of the computation (note that $G^{v_i^0}, G^{v_i^1}$ themselves will be copies of $G(y_{\leqslant k})$).
2. bit $i$ is **traversed** iff it is not known and there exist a pebble $P$ that visited the target of $G^{v_i^0}$ or $G^{v_i^1}$ of some copy of $G(y_{\leqslant k}, z)$ during the $(k+1)$-computation.
3. bit $i$ is **free** iff it is neither known nor traversed.

Finally, for a $(k+1)$-computation $C_1(Q, \Pi, z, L_{k+1})$ and a time $t$ during the computation, we define $KT_t(Q, \Pi, z, L_{k+1})$ as the set of bits that are known or traversed by the time $t$.

Similarly, we need an analogous definition of active pebble as in Definition 9.

**Definition 17.** We say a color $c$ is **active** at time $t$ during a $(k+1)$-computation if and only if one of the following holds at time $t$:

1. Some pebble of color $c$ is in the $G^{target}$ of some $G(y_{\leqslant k}, z)$.
2. Some pebble of color $c$ is in $G^u$ of some $G(y_{\leqslant k}, z)$ where $u$ is not a leaf.
3. Some pebble of color $c$ is in a $G^u$ where $u$ is a leaf of $G(y_{\leqslant k}, z)$ but the goal of $G^u$ is not occupied by any pebble of color $c$.

In words, we can think of an active color as a color with a "meaningful move" available to it (recall that colors cannot jump).

The following lemma is effectively the only new part of the proof: roughly speaking, it states that for traversing a copy $G^v$ of $G(y_{\leqslant k})$ (and not $G(y_{\leqslant k}, z)$ that we are most interested in), all pebbles of a color should appear inside $G^v$ (note that this lemma does not have a meaningful analogue in Lemma 5 beside saying that for a pebble to pass through a vertex, it should be on the vertex). In other words, $J$ needs to "dedicate" an entire color for traversing a copy of $G(y_{\leqslant k})$ (roughly because $G(y_{\leqslant k})$ is a "hard to traverse" graph by the choice of $y_{\leqslant k}$ in the induction step); we can then use this exactly as before by showing that to traverse a copy of $G(y_{\leqslant k})$, $J$ needs to "risk" losing an entire color to a leaf-copy of $G(y_{\leqslant k})$ and thus cannot traverse some $G(y_{\leqslant k}, z)$ during a $(k+1)$-computation.

**Lemma 18.** *For any $(Q, \Pi)$ and any $z \in \{0,1\}^d$, $x_{\geqslant k+2} \in \{0,1\}^{d\cdot(p-k-1)}$, and $L_{k+1} \in \{0,1\}^p$ such that $C_{k+1}(Q, \Pi, z, L_{k+1})$ traverses a copy $G^u$ of $G(y_{\leqslant k})$ at time $t$ with some color $c$. If there are $\geqslant p-k$ distinct colors according to $\text{color}^{\Pi}$ at time $t$, then* all *pebbles colored $c$, i.e., all $i \in [p]$ with $\text{color}^{\Pi}_t(i) = c$, are inside $G^u$ at time $t$.*

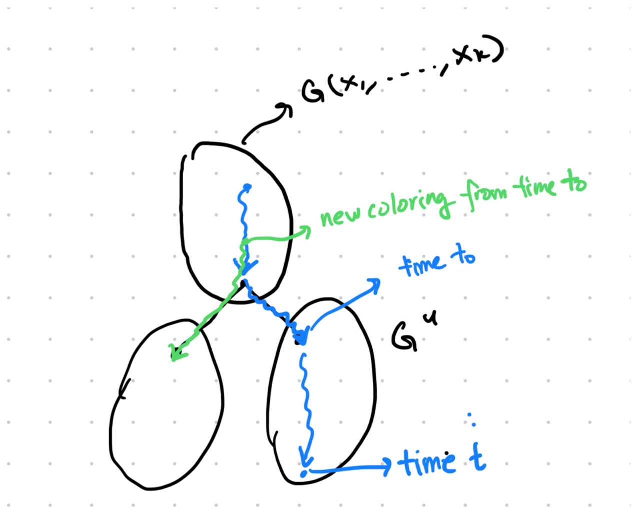

Figure 5: An illustration of the proof of Lemma 18.

*Proof.* Suppose towards a contradiction that there is a pebble $P_i$ such that $\text{color}^{\Pi}_t(i) = c$ but $P_i$ is not in $G^u$ at time $t$. Then at each time between 0 and $t$, there must be one pebble of color $c$ that is not in $G^u$. This is because color $c$ could have not jumped and if at some point all its pebbles were inside $G^u$, then until they traverse $G^u$, namely at time $t$, it could not have a pebble outside $G^u$ again.

Additionally, there must be a time $t_0$ such that there is a pebble of color $c$ on the start of $G^u$ and between $t_0$ and $t$ there is always one pebble of $c$ in $G^u$ (as eventually this color traversed the entire $G^u$). Let $(Q_0, \Pi_0)$

be the states and pebbles of $J$ at time $t_0$. Consider a *new* coloring $\text{color}^{\Pi_0}(\cdot)$ with respect to $\Pi_0$ (instead of $\Pi$ that defines $\text{color}^{\Pi}(\cdot)$). We claim that there must be at least $p-k+1$ distinct colors in this new coloring: firstly, since we are talking about a $(k+1)$-computation, throughout time $t_0$ to $t$, there are at least $p-(k+1)+1=p-k$ distinct colors in $\text{color}^{\Pi}(\cdot)$. At the same time, in $\text{color}^{\Pi_0}(\cdot)$, we will color the pebbles of color $\text{color}^{\Pi}(\cdot)=c$ with at least two different colors (because they are not all sitting on the same pebble, with one being outside $G^u$ and one being inside $G^u$ between $t_0$ to $t$). Thus, $\text{color}^{\Pi_0}(\cdot)$ has $p-k+1$ distinct colors between $t_0$ to $t$.

But now we have a contradiction: starting from $(Q_0,\Pi_0)$, between $t_0$ to $t$, we have $J$ going through a $k$-computation with $p-k+1$ distinct colors that traverses a copy of $G(y_{\leqslant k})$. This contradicts to the choice of $y_{\leqslant k}$ and Lemma 14 for $k$. Hence, the pebble $P_i$ could not exist in the same place and the lemma holds. □

The rest of the proof of Lemma 14 is almost verbatim as in Lemma 5. We have the following analogue of Lemma 10 for $(k+1)$-computations. The function $f(\cdot,\cdot)$ in the following is as defined before.

**Lemma 19** (Counting Lemma (II)). *Let $(Q,\Pi,L)$ be arbitrary, $d'\in\{0,1,\cdots,d\}$ and $p'\in\{0,1,\cdots,p\}$, and $t\geqslant 1$. Fix a set $S\subseteq\{0,1\}^d$ and suppose:*

1. *for all $z\in S$, $KT_t(Q,\Pi,z,L_{k+1})$ is the same set $B\subseteq[d]$;*
2. *$S$ is fixed on $B$; and,*
3. *$C_{k+1}(Q,\Pi,z,L_{k+1})$ has $d'$ free bits and at most $p'$ active colors for all $z\in S$.*

*Then,*

$$\#\text{ of } z\in S \text{ such that } C_{k+1}(Q,\Pi,z,L_{k+1}) \text{ traverses} \leqslant f(d',p').$$

*Proof sketch.* The proof is almost identical to Lemma 10 and we only mention the minor difference. The difference is that we would need to use Lemma 18 to argue that in the first time $t'$ after $t$ where a free bit flipped into traversed, there are $\leqslant p'-1$ active colors in time $t'$ for $S_1$. This is because by Lemma 18, all pebbles of this color are now stuck in that copy of $G(y_{\leqslant k})$ and that copy is a leaf-copy (so no more meaningful move) which means the number of active colors has to decrease by at least one. □

The proof of Lemma 14 is now exactly as in the last step of the proof of Lemma 5 and we omit it.

**Remark.** At a high level, the induction proof starts by considering when the pebbles are fully scattered which is the base case. Here the counting lemma shows that there is too many ways to construct the skinny trees that if you only have $p$ pebbles and $q$ states and never make a jump (thus pebbles are still fully scattered), your deterministic NNJAG will not be able to reach the goal of even the innermost layer of the skinny tree.

The number of distinct colors is then introduced as some measure of how scattered the pebbles are. By the restriction of this model, once a pebble is stuck at a dead-end the only thing it could do is to jump to another pebble. When it jumps, the pebbles become less scattered. Lemma 5 shows that it is not possible to traverse to the target of any copy of $G(x_1)$ without making a jump. So now the number of distinct colors has to decrease by 1.

Then Lemma 18 says that actually if you are able to traverse to the target of any $G(x_1)$ after that jump, all pebbles of that color must be inside that $G(x_1)$. Why? If that statement is not true, then the pebbles are too scattered and you are back to the case where such traversal is proved to not be possible. Repeating this argument completes the proof.

## References

[EPA99] Jeff Edmonds, Chung Keung Poon, and Dimitris Achlioptas. Tight lower bounds for st-connectivity on the NNJAG model. *SIAM J. Comput.*, 28(6):2257–2284, 1999. 2

[Poo93] C.K. Poon. Space bounds for graph connectivity problems on node-named jags and node-ordered jags. In *Proceedings of 1993 IEEE 34th Annual Foundations of Computer Science*, pages 218–227, 1993. 1, 2, 3