Power of Nondeterministic JAGs on Cayley graphs*

Martin Hofmann and Ramyaa Ramyaa

Ludwig-Maximilians Universität München
Oettingenstraße 67, 80538 Munich, Germany
{mhofmann,ramyaa}@tcs.ifi.lmu.de

Abstract. The Immerman-Szelepcsényi Theorem uses an algorithm for co-st-connectivity based on inductive counting to prove that NLOGSPACE is closed under complementation. We want to investigate whether counting is necessary for this theorem to hold. Concretely, we show that Nondeterministic Jumping Graph Automata (ND-JAGs) (pebble automata on graphs), on several families of Cayley graphs, are equal in power to nondeterministic logspace Turing machines that are given such graphs as a linear encoding. In particular, it follows that ND-JAGs can solve co-st-connectivity on those graphs. This came as a surprise since Cook and Rackoff showed that deterministic JAGs cannot solve st-connectivity on many Cayley graphs due to their high self-similarity (every neighbourhood looks the same). Thus, our results show that on these graphs, nondeterminism provably adds computational power.

The families of Cayley graphs we consider include Cayley graphs of abelian groups and of all finite simple groups irrespective of how they are presented and graphs corresponding to groups generated by various product constructions, including iterated ones.

We remark that assessing the precise power of nondeterministic JAGs and in particular whether they can solve co-st-connectivity on arbitrary graphs is left as an open problem by Edmonds, Poon and Achlioptas. Our results suggest a positive answer to this question and in particular considerably limit the search space for a potential counterexample.

1 Introduction

The technique of inductive counting forces nondeterministic machines to enumerate all possible sequences of nondeterministic choices. This is used in the Immerman-Szelepcsényi theorem (9) proving that nondeterministic space is closed under complementation and in (13) to show similar results for various space-complexity classes. This raises the question whether the operation of counting is needed for such exhaustive enumeration or indeed to show the closure under complementation of the space classes. To investigate this question for NLOGSPACE, we explore the problem of whether counting is needed to solve co-st-connectivity, i.e., the problem of determining that there is no path between two specified vertices of a given graph. Note that for nondeterministic machines such as our Pebble automata this is not trivially equivalent to st-connectivity (existence of a path) due to their asymmetric acceptance condition. For nondeterministic

* This work was supported by Deutsche Forschungsgemeinschaft (DFG) under grant PURPLE.
Turing machines with logarithmic space bound whose input (e.g. a graph) is presented in binary encoding on a tape, we know of course by Immerman Szelepcsenyi’s result that the two problems are equivalent. By \( \text{NLOGSPACE} \) we mean as usual the problems solvable by such a machine; thus, more generally, \( \text{NLOGSPACE} = \text{co-NLOGSPACE} \) by Immerman-Szelepcsenyi.

The framework to study the precise role of counting in that argument should be a formal system which cannot a priori count, but when augmented with counting, captures all of \( \text{NLOGSPACE} \). There are many natural systems operating over graphs which provably cannot count, e.g. transitive closure logic (TC) ([5]).

These systems get abstract graphs (not their encoding) as input and perform graph based operations such as moving pebbles along an edge. The main difference between such systems and Turing machines are that the latter get as input graph encodings, which embodies an ordering over the nodes. A total ordering over the nodes of a graph can be used to simulate counting. The system getting abstract graphs as inputs can still simulate counters up to certain values depending on the structure of the input graph.

In the case where a graph is presented as a binary relation it has been shown that positive TC (first order logic with the transitive closure operator used only in positive places within any formula) cannot solve co-st-connectivity ([6]). In the, arguably more natural, case where the edges adjacent to a node are ordered no such result is known. It is also to this situation that the aforementioned open question applies.

We consider here the simplest machines operating on such edge-ordered graphs - the nondeterministic version of jumping automata on graphs (JAGs).

Cook and Rackoff [2] introduced jumping automata on graphs JAGs (a finite automaton which can move a fixed number of pebbles along edges of a graph) and showed that they cannot solve st-connectivity on undirected graphs (ustcon). It also known [10] that JAGs augmented with counters, called RAMJAGs, can simulate Reingold’s algorithm [12] for ustcon. Among the consequences of this are that JAGs only need the addition of counters to capture all of \( \text{LOGSPACE} \) on connected graphs and that without the external addition of counters, JAGs cannot count. Thus, ND-JAGs may provide us with a platform to explore whether counting is needed for co-st-connectivity and more generally all of \( \text{NLOGSPACE} \). While nondeterministic JAGs (ND-JAGs) can obviously solve st-connectivity, even for directed graphs, it is unclear whether they can solve co-st-connectivity. Note that if an ND-JAG, on an input directed graph, can order the graph when the edge directions are disregarded, one could order the underlying undirected graph and use that order to simulate Immerman-Szelepcsenyi so as to decide co-st-connectivity for the directed graph. So, nothing is lost by restricting our investigation to undirected graphs.

Several extensions of JAGs such as JAGs working on graphs with named nodes ([11]), probabilistic, randomized and nondeterministic JAGs ([1], [11], [4]) have been studied, but with the motivation of establishing space/time lower or upper bounds of solving reachability questions as measured by the number of states as a function of size and degree of the input graph. This differs from our motivation.

A non-constant lower bound in this sense for solving co-st-connectivity with nd-jags would imply that counting is needed, but we are not aware of space lower bounds for ND-JAGs.
The question of implementing Immerman-Szelepcsenyi’s algorithm in ND-JAGs is considered in [4], but has been left as an open question.

The results of this paper constitute an important step towards the solution of this question. Namely, we show that ND-JAGs can order and thus implement any NLOGSPACE-algorithm on a variety of graph families derived from Cayley graphs. This came as a surprise to us because Cayley graphs were used by Cook and Rackoff to show that deterministic JAGs cannot solve st-connectivity. Intuitively, this is due to the high degree of self-similarity of these graphs (every neighbourhood looks the same). It thus seemed natural to conjecture, as we did, that ND-JAGs cannot systematically explore such graphs either and tried to prove this. In the course of this attempt we found the conjecture to be false and discovered that in the case of pebble automata, nondeterminism indeed adds power.

Our main results show that the Cayley graphs of the following groups can be ordered by ND-JAGs: all abelian groups and simple finite groups irrespective of the choice of generators, (iterated) wreath products $G \triangleleft H$ once $G$ can be ordered and a mild size condition on $H$ holds. This implies that none of these groups can serve as a counterexample for a possible proof that ND-JAGs cannot solve co-st-connectivity on undirected graphs.

One may criticize that we do not in this paper answer the question whether or not ND-JAGs can solve co-st-connectivity on undirected graphs. However, this is not at all an easy question and we believe that our constructions will help to eventually solve it because they allow one to discard many seemingly reasonable candidates for counterexamples such as the original counterexamples from Cook and Rackoff or the iterated wreath products thereof that were used by one of us and U. Schöpp [8] in an extension of Cook and Rackoff’s result.

An anonymous reviewer of an earlier version wrote that trying to argue that ND-JAGs can do co-st-conn without counting “seems silly”. One should “just take a graph that is between structured and unstructured and then it likely can’t be done.”. When we started this work this would have been our immediate reaction, but on the one hand, it is not easy to construct a graph “between structured and unstructured” in such a way that one can prove something about it. On the other hand, we found the additional power of nondeterminism in this context so surprising that we are no longer convinced that ND-JAGs really cannot do co-st-connectivity and hope that our constructions will be helpful in the process of deciding this question.

2 Preliminaries/definitions

2.1 Jumping Automata on Graphs

Cook and Rackoff ([2]) introduced Jumping Automata on Graphs (JAGs) in order to study space lower bounds for reachability problems. A JAG is a finite automaton with a fixed number of pebbles and a transition table. It gets as input a graph of fixed degree and labeled edges and begins its computation with all of its pebbles placed on a distinguished start node. During the course of its computation, the JAG moves the pebbles (a pebble may be moved along an edge of the node it is on, or to the node that has another
A labelled degree $d$ graph for $d > 1$ comprises a set $V$ of vertices and a function $\rho : V \times \{1, \ldots, d\} \to V$. If $\rho(v, i) = v'$ then we say that $(v, v')$ is an edge labelled $i$ from $v$ to $v'$. All graphs considered in this paper are labelled degree $d$ graphs for some $d$. The important difference to the more standard graphs of the form $G = (V, E)$ where $E \subseteq V \times V$ is that the out degree of each vertex is exactly $d$ and, more importantly, that the edges emanating from any one node are linearly ordered. One extends $\rho$ naturally to sequences of labels (from $\{1, \ldots, d\}^*$) and writes $v' = v, w$ if $\rho(v, w) = (v', v, v' \in V$ and $w \in \{1, \ldots, d\}^*$. The induced sequence of intermediate vertices (including $v, v'$) is called the path labelled $w$ from $v$ to $v'$. Such a graph is undirected if for each edge there is one in the opposite direction $(\rho(v, i) = v' \Rightarrow \exists j, \rho(v', j) = i$. It is technically useful to slightly generalise this and also regard such graphs as undirected if each edge can be reversed by a path of a fixed maximum length.

Definition 1. A $d$-Jumping Automaton for Graphs (d-JAG), $J$, consists of

- a finite set $Q$ of states with distinguished start state $q_0$ and accept state $q_a$
- a finite set $P$ of objects called pebbles (numbered 1 through $p$)
- a transition function $\delta$ which assigns to each state $q$ and each equivalence relation $\pi$ on $P$ (representing incidence of pebbles) a set of pairs $(q', c)$ where $q' \in Q$ is the successor state and where $c = (c_1, \ldots, c_p)$ is a sequence of moves, one for each pebble. Such a move can either be of the form $\text{move}(i)$ where $i \in \{1, \ldots, d\}$ (move the pebble along edge $i$) or $\text{jump}(j)$ where $j \in \{1, \ldots, p\}$ (jump the pebble to the (old) position of pebble $j$).
- The automaton is deterministic if $\delta(q, \pi)$ is a singleton set for each $q, \pi$.

The input to a d-JAG is a labelled degree $d$ graph. An instantaneous description (id) of a d-JAG $J$ on an input graph $G$ is specified by a state $q$ and a function, node, from the $P$ to the nodes of $G$ where for any pebble $p$, $\text{node}(p)$ gives the node on which the pebble $p$ is placed.

Given an id $(q, \text{node})$ a legal move of $J$ is an element $(q', c) \in \delta(q, \pi)$ where $\pi$ is the equivalence relation given by $p \pi p' \iff \text{node}(p) = \text{node}(p')$.

The action of a d-JAG, or the next move is given by its transition function and consists of the control passing to a new state after moving each pebble $i$ according to $c_i$: (a) if $c_i = \text{move}(j)$ move $i$ along edge $j$; (b) if $c_i = \text{jump}(j)$ move (jump) it to $\text{node}(j)$.

Any sequence (finite or infinite) of ids of a d-JAG $J$ on an input $G$ which form consecutive legal moves of $J$ is called a computation of $J$ on $G$. We assume that input graphs $G$ have distinguished nodes $\text{startnode}$ and $\text{targetnode}$, and that d-JAGs have dedicated pebbles $s$ and $t$. The initial id of $J$ on input $G$ has state $q_0$, $\text{node}(t) = \text{targetnode}$ and $\forall q \neq t. \text{node}(q) = \text{startnode}$. $J$ accepts $G$ if the computation of $J$ on $G$ starting with the initial id ends in an id with state $q_a$.

Connected components A d-JAG cannot count, and so, is a very weak model over discrete graphs (graphs with no edges). Since we are interested in investigating co-st-
connectivity, this behaviour on graphs with a large number of components is not relevant. To avoid such trivialities we assume that graphs always have at most two connected components and that each connected component is initially pebbled with either \( s \) or \( t \). Thus, there is initially no connected component without pebbles.

As already mentioned, jags are nonuniform with respect to the degree. This can be overcome by modifying the definition to place the pebbles on the edges instead of nodes, and modifying the pebble-move function to use the edge order instead of outputting a specified labeled edge. Concretely, a pebble placed on an edge can either be moved to the next edge coming out of the current edges source (called the \textit{next} move) or to the first edge coming out of the current edges target (called the \textit{first} move).

Alternatively, we can assume that a degree-\( d \) graph is first transformed to a degree-3 graph by replacing each node with a cycle of size \( d \) (replacement product). In the sequel, when running a JAG or ND-JAG on a family of graphs where the degree is not known in advance, we always assume that one of these modifications has been put in place.

Cook and Rackoff’s result [2] shows that even with this modifications, JAGs can only compute local properties.

Theorem 1 ([2]). (u)st-connectivity cannot be solved by JAGs.

2.1.1 Counters

Since counting is central to our study, we present various results pertaining to the ability of JAGs to count. It is obvious that JAGs cannot count the number of nodes of a discrete graph.

The following Lemma is a direct consequence of Cook-Rackoff and Reingold’s result.

Lemma 1. JAGs cannot count the number of nodes of connected graphs.

Proof. JAGs cannot explore connected graphs ([2]), while JAGs augmented with counters can implement Reingold’s algorithm for st-connectivity [10].

On the other hand, on particular graphs, JAGs do have a certain ability to count. For instance, if two pebbles are placed on nodes connected by a (non self-intersecting) path of identically labelled (say 2) edges then the pebbles can be interpreted as a counter storing a value up to the number of edges in this path. Any implementation of counters by a deterministic JAG has to count by repeating some fixed sequence of moves (in the above example, this was ‘move along edge 2’). The result by [2] for the deterministic JAGs used graphs with low “exponent” (the maximum number of times any series of moves can be repeated without looping), so that JAGs on such graphs cannot count to a high value. This situation changes when we add non-determinism.

It is well-known that JAGs can implement any LOGSPACE algorithm on connected graphs with a total order (made available in an appropriate way, e.g. by providing a path with a specific label that threads all nodes, or by an auxiliary JAG that is able to place a pebble on one node after the other). A consequence of the implementability of Reingold’s algorithm on JAGs equipped with counters is that those can order connected edge-labelled graphs and so can implement any LOGSPACE algorithm on such graphs (a run of Reingold’s algorithm induces a total order on the nodes). This shows that this model is quite powerful.
2.2 Nondeterministic Jumping Automata on Graphs

Definition 2. A nondeterministic Jumping Automaton for Graphs (ND-JAG) $J$ is a JAG whose transition function is nondeterministic. It accepts an input if there is some finite computation starting at the initial configuration that reaches $q_a$, and rejects an input if no such computation does. A $d$–ND-JAG operates on graphs of degree $d$. Again, we assume that appropriate degree reduction is applied before inputting a graph to an ND-JAG.

It is easy to see that the argument used in [2] for deterministic JAGs which is similar to a pumping argument cannot be adapted easily to ND-JAGs, which can solve reachability (guess a path from startnode to targetnode). However, it is unclear whether ND-JAGs can solve co-st-connectivity. Since JAGs cannot count, it is reasonable to believe that ND-JAGs cannot implement Immerman-Szelepcsenyi’s algorithm, and more generally, cannot solve co-st-connectivity.

2.3 Cayley graphs

Cayley graphs encode the abstract structure of groups. The Cayley graph of the group $G$ with generators $m$, written as $CG(G, m)$ is the labelled degree $|m|$ graph whose nodes are the elements of $G$ and where the edge labelled $i$ from node $v$ leads to $m_i v$, formally $\rho(v, i) = m_i v$.

We take the node corresponding to $1_G$ to be startnode.

We note that Cayley graphs are “undirected” in the relaxed sense since for every edge there is a fixed length path (labelled by the inverse of the generator labelling the edge) that inverts it. Indeed, connectivity and strong connectivity coincide for them. Often, the definition of Cayley graphs presupposes that the set of generators be closed under inverses in which case they are undirected in the strict sense.

It follows directly that (1) st-connectivity for Cayley graphs is in LOGSPACE. (2) For any two nodes $u$ and $v$, there is an automorphism that maps $u$ to $v$ (3) For any two pairs of nodes $(u_1, v_1)$ and $(u_2, v_2)$, either every sequence of labeled edges that reaches $v_1$ from $u_1$ also reaches $v_2$ from $u_2$, or none does. So we give paths as a list of edge-labels and define a function target such that $\text{target}(u, p) = v$ iff $p$ is a path between $u$ and $v$. We refer to the list obtained by appending $b$ to $a$ by either $ab$ or $a : b$. (4) For any two nodes $u$ and $v$, there is exactly one $w$ such that for every path $\rho$ from $u$ to $v$, $\rho$ is a path from $w$ to $u$.

3 Algorithms

Theorem 2. ND-JAGs on Cayley graphs can perform group multiplication and inverse.

Proof. Multiplication: Given $p$ and $q$, to place $r$ on node($p$).node($q$) proceed as follows: Jump pebble $p'$ to $s$ (start node) and $r$ to $q$. Nondeterministically, trace a path from $s$ to $p$ using $p'$. In tandem with the moves made by $p'$, move $r$. Halt when $p'$ reaches $p$. Inverse: Given $p$ to place $q$ on a node such that node($q$).node($p$) = node($s$) proceed as follows: Nondeterministically move the pebble $q$ such that node($q$).node($p$) = node($s$).
ND-JAGs can perform other operations such as verifying that the input is not a Cayley graph, determining if a node is an element of the center, determining if the subgroup generated by given elements is normal, etc. However, it is not obvious whether several related operations, such as checking whether the input is a Cayley graph, are doable.

If an ND-JAG can be guaranteed to place a dedicated pebble on all the nodes of a class of input graph that are connected to the start node, then it follows that ND-JAGs can solve co-st-connectivity on this class of graphs. We call such graphs to be traversable. However, co-NLOGSPACE may require a stronger property. It is well-known that if an ND-JAG can order a class of graphs (i.e., can compute a total order over the nodes) then it can count, and as a consequent, decide any (co)-NLOGSPACE property over the class of graphs. We call such graphs orderable.

Definition 3. A family of graphs is traversable if there is some ND-JAG J such that for each graph G in the family all accepting computations involve placing a dedicated pebble curr on every node reachable from startnode, and J does accept G (i.e. there is at least one accepting computation). The family is orderable if it is traversable by an ND-JAG J, and for every accepting computation of J the ordered sequence of nodes on which curr is placed along the computation is always the same. A family of groups with generators is traversable (or orderable) if its Cayley graphs are.

Theorem 3. If a family of graphs is traversable, ND-JAGs can solve co-st-connectivity on it. If a family of graphs is orderable then any ND-JAGs can decide any NLOGSPACE property on the subgraphs reachable from startnode.

Proof (sketch). To decide co-st-connectivity simply traverse the graph and see whether curr ever reaches targetnode. To decide an NLOGSPACE-property first note that by repeatedly cycling curr through the graph (the component reachable from startnode to be precise) we can count up to its size. This allows us to simulate logarithmically sized worktapes as counters. Since the graph is not merely traversable but orderable we can define an encoding of graph nodes as numbers thus allowing us to simulate any NLOGSPACE Turing machine.

3.1 Grid Graphs

Let the family of groups grid(d, ℓ) indexed by d, ℓ ∈ N be (Z/ℓZ). So grid(d, ℓ) has d commuting generators (say, m_1, . . . , m_d) each of order ℓ. Cook and Rackoff [2] showed that deterministic d-JAGs are unable to traverse these graphs and thus were able to conclude that deterministic JAGs cannot in general decide st-connectivity for undirected graphs.

We show that CG(grid(d, ℓ)) is traversable. Every node in CG(grid(d, ℓ)) can be reached from startnode by a unique path of the form m_i_1 : · · · : m_i_d where 0 ≤ i_j < ℓ. To ensure that curr visits every node, retrace nondeterministically a path ρ of that form to the current position curr using a helper pebble x. In tandem with the helper pebble moving towards curr move another helper pebble y along the path that is lexicographically the successor of ρ. If ρ = m_i_1 : · · · : m_i_d where, (0 ≤ i_j < ℓ) ∧ (i_k < ℓ − 1) ∧ (∃k.∀j > k. i_j = ℓ − 1)
then the lexicographically next path $\rho' = m_{i_1}^{i_1} : \cdots : m_{i_k}^{i_k + 1}$. Then jump $curr$ to the final position of $y$.

The following algorithm places the pebble $curr$ on every node of the input graph $CG(\text{grid}(d, \ell))$. We also use pebbles travelling around to store directions $k$ and $dd$.

pebbles: $s$, curtrace, curr, next, count (all at startnode)
direction: $k$, $dd$

repeat
  %% find the next node to move curr
  nondeterministically guess $k$:{1..d}
  for $dd = 1$ to $k$
    count := $s$
    nondeterministically guess $b$:{true,false}
    while ($b$)
      nondeterministically guess $b$:{true,false}
      next := next.$dd$
      count := count.$dd$
      if (count == $s$) fail  %% checks that $i_{j} < \ell$
      if (count.$k$ == $s$) fail  %% checks that $i_{k} < \ell - 1$

  %% check that the "immediate" next node has been found
  curtrace := next
  for $dd = k$ to $md$  %% for $j > k$, $i_{j} = \ell - 1$
    count := $s$.dd
    while (count != $s$)
      curtrace := curtrace.$dd$
      count := count.$dd$
      if (curtrace != curr) fail
      curr := next
  while (curr != $s$)

Corollary 1. ND-JAGs can simulate any NLOGSPACE algorithm on $CG(\text{grid}(d, \ell))$.

The algorithm is based on the fact that between any two nodes, there exists a unique, verifiable path $\rho$ of the form $\rho = m_{i_1}^{i_1} : \cdots : m_{i_k}^{i_k}$ where, $(0 \leq i_j < \ell) \land (i_k < \ell - 1) \land (\exists k. \forall j > k. i_j = \ell - 1)$, and that between paths of this form, we can define a verifiable total order (the lexicographic order). Using this notion of verifiable and orderable canonical paths, we can generalize this algorithm to other graphs.

A predicate $R$ over paths of a graph is verifiable if given nodes $u$, $v$, and $w$, an ND-JAG can determine if there is a path $\rho$ from $u$ to $v$ with $R(\rho)$ via $w$. A total order $O$ over such a predicate $R$ over paths is verifiable if given nodes $u$, $v$, and $w$, an ND-JAG can determine if some path $\rho'$ in $R$ from $u$ is passes through $w$ and $\rho'$ is the next path
of $\rho$ by $O$ where $\text{target}(u, \rho) = v$. Given a graph $G$, define a predicate canonical path over paths such that it is verifiable, and for any node $u$, there is exactly one canonical path from startnode to $u$; and define a verifiable total order over canonical paths. Given any node, an ND-JAG can trace the canonical path (from startnode) to it, as well as the next canonical path. $G$ is then orderable by repeating this.

In the rest of this paper we will generalise this method first to Cayley graphs of arbitrary abelian groups and subsequently to selected non-abelian groups.

### 3.2 Abelian groups

Here, we show that Cayley graphs of abelian groups no matter how presented can be ordered by an ND-JAG. Thus, let $G$ be an abelian group with generators $g_1, \ldots, g_d$. Let $e$ be the maximum order of the $g_i$. If $X \subseteq G$ we denote $\langle X \rangle$ the subgroup generated by $X$.

Let $\text{poly}$ stand for a fixed but arbitrary polynomial.

**Lemma 2.** An ND-JAG can count till $\text{poly}(e)$.

*Proof.* By scanning through the generators, identify the one with the maximum order, say $g_m$, and keep a pebble there. Clearly, by moving a pebble along the direction corresponding to $g_m$ one can count till $e$. Doing the same with more pebbles we can then count till $\text{poly}(e)$.

**Definition 4.** For each $i$, let $e_i$ be the order of $g_i$ in the factor group $G/\langle g_1, \ldots, g_{i-1} \rangle$. So, $e_1$ is the order of $g_1$ in $G$, $e_2$ is the least $t$ so that $g_2^t$ can be expressed in terms of $g_1$; $e_3$ is the least $t$ so that $g_3^t$ can be expressed in terms of $g_1$ and $g_2$.

A path $w = g_1^{t_1}, \ldots, g_n^{t_n}$ is canonical if each $t_i < e_i$.

**Lemma 3.** For every element of $G$ there is a unique canonical path reaching it.

*Proof.* Clearly, every element of $G$ can be reached by a canonical path (“canonize” from $g_n$ downward). As for uniqueness suppose that $g_1^{t_1}, \ldots, g_n^{t_n} = g_1^{u_1}, \ldots, g_n^{u_n}$ and $t_i, u_i < e_i$. Dividing by the RHS using commutativity and reducing mod $e_i$ we obtain $g_1^{t_1}, \ldots, g_m^{t_m} = e$ (for different $t_i < e_i$) where $m \leq n$ and $t_m \neq 0$. This, however, implies that $g_m^{t_m}$ can be expressed in terms of $g_1, \ldots, g_{m-1}$, a contradiction.

**Definition 5.** For each $1 \leq i \leq n$ define $\text{gmax}_i$ as $g_1^{e_1-1} \cdots g_n^{e_i-1}$ and $\text{nmax}_i$ as the length of the canonical path leading to $\text{gmax}_i$, i.e., $\text{nmax}_i = e_1 - 1 + \cdots + e_i - 1$.

**Lemma 4.** Let $w = g_1^{t_1}, \ldots, g_i^{t_i}$ be a path reaching $\text{gmax}_i$ and of total length $\text{nmax}_i$, i.e., $t_1 + \cdots + t_i = \text{nmax}_i$. Then $w$ is the canonical path to $\text{gmax}_i$ and thus $t_j = e_j - 1$ for all $j$. 

Proof. By uniqueness of canonical paths the exponents $t_i$ are uniquely determined mod $e_i$. The claim follows directly from that.

Lemma 5. Given $nmax_i$ and $gmax_i$, we can reconstruct $nmax_j$ and $gmax_j$ for all $j < i$.

Proof. We nondeterministically guess a path of the form $g_1^*, \ldots, g_i^*$ to $gmax_i$ and check that its length is indeed $nmax_i$. On its way it passes through $gmax_j$ for $j < i$ allowing us to read off all intermediate values $nmax_j$.

Lemma 6. Given $nmax_i$ and $gmax_i$ we can enumerate the subgroup generated by $g_1, \ldots, g_i$.

Proof. Using the previous lemma we can check whether a path that remains in this subgroup is canonical. This allows us to enumerate the subgroup as in the case of grid graphs described earlier.

Lemma 7. Given $nmax_i$ and $gmax_i$, we can compute $nmax_{i+1}$ and $gmax_{i+1}$.

Proof. Compute $x_t := g_{i+1}^t$ for $t = 1, 2, \ldots$ and after each step check using the previous lemma determine the least $t$ so that $x_t$ is in the subgroup generated by $g_1, \ldots, g_i$. Then $gmax_{i+1} = x_{i-1}$ and $nmax_{i+1} = nmax_i + t - 1$.

The following is now direct.

Theorem 4. Any abelian group can be traversed by an ND-JAG.

3.3 Families of groups

Symmetric Group: Let $S(n)$ be the symmetric group of permutations of $1, \ldots, n$. This is generated by the generators $< cy, sw >$ where $sw$ is the permutation $(1, 2)$ and $cy$ is the permutation $(2, 3, \ldots, n, 1)$.

Canonical path: In $CG(S(n), cy, sw)$, every node can be reached from startnode via a path of the form $p_n^i, p_{n-1}^i \ldots p_2^i$ where $0 < j_i < n$ and $p_k = cy : (sw : cy)^{n-k} : cy^{k-1}$. ($p_k$ is the permutation $(n-k+1, n-k+1, \ldots, n)$). Any permutation can be performed by repeating $p_1$ to place the desired element is in the first position, repeating $p_2$ to place the desired element in the second position and so on.)

General linear group: The group of invertible matrices over finite fields with matrix multiplication as the group action. We take as generators the matrices that perform the following operations: $(\omega)$ multiplying the first column with the primitive element $\omega$ of the field; $(c_{1+2})$ adding the first column to the second; $(c_{12})$ permuting the first two columns; $(c_{cy})$ rotating the columns.

Every invertible matrix can be transformed into a diagonal matrix with nonzero elements using row and column transformations. Thus every element in this group can be
systematically generated by generating all diagonal matrices, and for each one generated, perform each possible column transformations, and for each such transformation, perform each possible row transformations. This gives a canonical path if the transformations and diagonal matrices can be ordered.

Every column transformation (any permutation of the columns, multiplying any column with a field element, adding the first column to another column) can be systematically generated using these generators (as in symmetric groups) using the fact that all nonzero elements of a field are generated by repeated multiplication of the primitive element; row transformations can be simulated using column transformations (if \( M \) is the operator for a particular column transformation, and the corresponding row transformation of matrix \( A \) can be performed by \( A.M \)); diagonal matrices can be generated using repeated multiplication with the primitive element and row and column permutations.

**Simple finite groups:**

Similar presentations exist for the other finite simple groups which allows their Cayley graphs to be ordered in an analogous fashion assuming a given presentation.

Furthermore, Guralnick et al.\[7\] have shown that all nonabelian simple finite groups that are not Ree groups, can be written using 2 generators and a constant number of relations. By embodying hard-coded versions of these relations it is possible to design a fixed ND-JAG that can order any Cayley graph whose underlying group is one of those to which loc.cit. applies, irrespective of the presentation. Essentially, the ND-JAG would nondeterministically guess the two generators and try out all of the hard-coded relations. Details will be given in an extended version of this paper.

Unfortunately, although any finite group can be decomposed into simple groups, it remains unclear how to use this, in order to traverse or order arbitrary Cayley graphs without prior knowledge about the underlying groups.

### 3.4 Product constructions

**Theorem 5.** If groups \( G \) and \( H \) are orderable, so are their direct and semidirect products. If graphs \( G \) and \( H \) are orderable, (and are compatible) so are their replacement product and zigzag product. Moreover, these constructions are uniform in \( G, H \) thus lift to families.

**Proof.** The direct product of \( G \) and \( H \) with (generators \( m \) and \( l \) resp.), \( G \times H \), has generators \( m, n \), elements of the form \((g, h)\) with \( g \in G \) and \( h \in H \), with \((g_1, h_1)(g_2, h_2) = (g_1 g_2, h_1 h_2)\). To order \( G \times H \), run the ND-JAG that ordered \( G \), and for each node \( g \in G \), run the ND-JAG that orders \( H \) (with \( s \) at \( g \)) to order the coset \( gH \) of \( H \). Semidirect product is similar.

For graphs \( G \) (with edges \( m \)) and \( H \) (with edges \( n \)) where \( |H| = \text{deg}(G) \), and an order \( O \) over \( H \), the replacement product of \( G \) and \( H \), \( G \otimes H \), has vertices \((g, h)\) with \( g \in G \) and \( h \in H \) and edges \( n_i \) with edge \( n_i \) from \((g_1, h_1)\) to \((g_1, h_2)\) if the edge \( n_i \) from \( h_1 \) leads to \( h_2 \) in \( H \), and edge \( r \) from \((g_1, h_i)\) to \((g_2, h_j)\) if the edge \( m_i \) from \( g_1 \) leads to \( g_2 \) in \( G \). Given that ND-JAGs can order \( G \) and \( H \) (according to \( O \)), to order \( G \otimes H \), modify the ND-JAG that orders \( G \) so that moves along edge \( m_i \) are simulated.
by traversing to the $i^{th}$ node of $H$ and moving along $r$. Further, at each node of $G$, traverse all the nodes of $H$ (before following the desired edge).\footnote{Converting a family of nonuniform $\text{ND-JAG}$s over graphs $L$ parameterized by degree $d$ to a uniform $\text{ND-JAG}$ uses $L(d)\otimes CG(Z/dZ)$.} Zigzag product can be traversed similarly.

Ordering of product groups using an $\text{ND-JAG}$ that orders its factor groups will result in an increase in the machine size. Hence, it is reasonable to assume that iterating such product constructions might not be orderable. For instance, the following construction, which iterates the wreath product construction was used in \cite{8} to construct a family of graphs over which the formal system \textsc{purple} cannot solve reachability.

\[ \Lambda(H, G, 1) = H; \]
\[ \Lambda(H, G, i + 1) = G \wr \Lambda(H, G, i) \]

Here, we show that if $G$ and $H$ are families of groups such that $G$ can be traversed / ordered, and $|H|$ can be computed from its number of generators by some register machine that does not store values greater than its output then so can the wreath product $G \wr H$ can be traversed / ordered, too. Note that we do not require that $H$ should be orderable/traversable. This shows iterated constructions such as the above one (or even modifications based on diagonalisation that achieve higher growth rates, e.g. Ackermann function) can be traversed / ordered.

\textit{Properties of wreath products.} Consider groups $G$ and $H$ with generators $m$ and $n$ respectively. Their wreath product, $G \wr H$ has $|G|^{|H|} \ast |H|$ elements of the form $(f, h)$ where $f : H \to G$ (not necessarily a homomorphism) and $h \in H$, with identity $(\lambda x. \text{id}_G, \text{id}_H)$, and multiplication operation defined as $(f_1, h_1)(f_2, h_2) = (\lambda h_3, f_1(h_3) f_2(h_1^{-1} h_3), h_1, h_2)$.

Both $G$ and $H$ embed faithfully into $G \wr H$ by $g \mapsto (\delta_g, 1_H)$ and $h \mapsto (\delta_1, h)$ where $\delta_1(h) = g$ and $\delta_g(h) = 1_G$ when $h \neq 1_H$. We identify elements of $G$ and $H$ with these embeddings where appropriate. The (embeddings of the) generators $G$ and $H$ together generate $G \wr H$ and we define those to be the distinguished generators of the wreath product.

If $h = h_1, \ldots, h_n$ are pairwise distinct elements from $H$ and $g = g_1, \ldots, g_n$ are (not necessarily distinct) elements from $G \setminus \{1_G\}$ we define $[h \to g] = (f, 1_H) \in G \wr H$ where $f(h_i) = g_i$ and $f(h) = 1_G$ when $h \not\in \{h_1, \ldots, h_n\}$.

We have
\[ [h \to g] = h_1 g_1 h_1^{-1} h_2 g_2 h_2^{-1} \ldots h_n g_n h_n^{-1} \]

(1)

\textit{Counting up to $|H|$.} We can now use elements of this form to represent numbers; to be precise we represent $n$ by elements $(f, 1_H)$ where $|[h \mid f(h) \neq 1_G]| = n$.

We now consider $\text{ND-JAGs}$ run on the Cayley graph of $G \wr H$ with the above generators. We assume a fixed start pebble allowing us to identify pebble positions with elements of $G \wr H$.

\textbf{Lemma 8.} We can test whether an element $x \in G \wr H$ represents a number.
Proof. Nondeterministically guess a path (from the fixed start pebble) to \( x \) and check that the \( H \)-generators contained in that path multiply to \( 1_H \).

**Lemma 9.** Given a number representation \( x = (f, 1_H) \) and an element \( h \in H \) (both by pebbles) it is possible to test whether \( f(h) = 1_G \).

**Proof.** Nondeterministically move a pebble \( a \) to \( x \) on a path \( \rho \). Use a distinguished pebble \( u \) to store the current \( H \)-value (the second component of \( a \)) as we go along the path. To do this, we apply the \( H \)-generators contained in the path \( \rho \) to \( u \). Another distinguished pebble \( v \) will represent the value \( f(h) \in G \) when \( a = (f, u) \). To do this, check at all times whether \( u \) happens to be equal to \( h \) and in that case apply \( G \)-generators in \( \rho \) to \( v \). When \( u \neq h \), let those \( G \)-generators pass. Then, when \( a \) finally reaches \( x \), we check whether \( v = 1_G \).

**Lemma 10.** Given a number representation \( x \) it is possible to nondeterministically guess a path to \( x \) that has the format in Equation 1.

**Proof.** We trace a path \( \rho \) to \( x \) and store the values \( h_1 \) and \( h_i \) in pebbles \( u \) and \( v \). Initially we set both \( u \) and \( v \) to the first \( H \)-block (maximal subword of \( H \)-generators) in \( \rho \). As we progress, we accumulate the next \( H \)-block in a pebble \( w \), compute \( h := vw \) and check using Lemma 9 that \( h \) is not among \( h_1, \ldots, h_i \). Should this happen, we fail. Otherwise, we can update \( v \) to \( h \) and continue.

**Lemma 11.** Given two number representations we can test whether they represent the same number and whether one represents the successor of the other.

**Proof.** This is a direct consequence of Lemma 10: trace canonical paths and check that their respective numbers of \( H \)-blocks are equal or in the successor relation.

**Lemma 12.** If \( |H| \) is computable from the number \( |n| \) of \( H \)'s generators by a register machine which never stores a value larger than its final output then an ND-JAG can count till \( |H| \).

**Proof.** The control of the register machine can be hard-coded in the transition table of an ND-JAG while number representations are used to encode registers. Register values are guaranteed never to exceed \( |H| \) and the operations of increment, decrement and checking for zero can be done as described in the above lemmas.

For example, if \( |H| \) is an iterated power of 2 then the premise of the lemma applies. The lemma is needed in the first place because the simulation of arithmetic contained in the earlier lemmas does not in general allow the detection of overflow and hence wraparound. Once we know \( |H| \) (e.g. by the simulation of a register machine which computes \( |H| \) from \( |n| \), never storing a value larger than \( |H| \)) we can then count till \( |H| \).
Theorem 6. Let $G$, $H$ be groups such that $G$ can be traversed(ordered) and such that the generators of $H$ are either given explicitly or can be computed by a single ND-JAG. Then there exists an ND-JAG traversing(ordering) the Cayley graph of $G \wr H$ and the construction is uniform in $G$, $H$.

Proof. The previous lemmas together with Theorem[12] show that we can perform arbitrary arithmetic operations with values up to $\text{poly}(|H|)$. This allows us to implement Reingold’s algorithm on $H(n)$ and hence traverse (order) $H$. It is then easy to traverse (order) $G \wr H$.

4 Conclusion

Algorithms for co-st-connectivity use techniques based on counting. We wanted to explore the question of whether the operation of counting is necessary to solve the problem of co-st-connectivity. For this purpose we considered jumping automata on graphs systems which cannot count, and consequently cannot do u-st-connectivity, but are powerful otherwise. This was the weakest of such models.

We added nondeterminism to JAGs, and studied the problem of coreachability on Cayley graphs which were used to show that deterministic JAGs cannot solve undirected reachability. This result exploited the rigid structure of such graphs - in particular, the fact that for any two nodes of a Cayley graphs, there is an automorphism that maps one to the other. To our surprise, we found that nondeterminism can exploit this property. Many families of Cayley graphs and product constructions over them can be ordered using ND-JAGs, and in consequence any (co)-NLOGSPACE algorithm can be implemented, including co-st-connectivity. Since this machine model is the weakest, these results hold for stronger models such as positive transitive closure logic as well.

While we are as yet unable to show that ND-JAGs can traverse arbitrary graphs, our results considerably restrict the search space for possible counterexamples. On the one hand, they should exhibit a very regular structure so that deterministic JAGs cannot already solve reachability; on the other hand, the very presence of such regular structure allowed the applicability of our new nondeterministic algorithms in all concrete cases.

References

1. P. Berman and Janos Simon. Lower bounds on graph threading by probabilistic machines. In Found. of Comp. Sci., 1983., 1983.
2. Stephen A. Cook and Charles Rackoff. Space lower bounds for maze threadability on restricted machines. SIAM J. Comput., 9(3):636–652, 1980.
3. Damm and Holzer. Inductive counting below logspace. In In Proc of the 19th Conf on Math. Found. of Comp. Sci., num 841 in LNCS. Springer, 1994.
4. J. Edmonds, C. Poon, and D. Achlioptas. Tight lower bounds for st-connectivity on the nnjag model. SIAM Journal on Computing, 28(6):2257–2284, 1999.
5. Kousha Etessami and Neil Immerman. Reachability and the power of local ordering. Th. Comp. Sci., 148(2):261–279, 1995.
6. Erich Grädel and Gregory L. McColm. Hierarchies in transitive closure logic, stratified datalog and infinitary logic. Ann. Pure Appl. Logic, 77(2):169–199, 1996.
7. Robert M Guralnick, William M Kantor, Martin Kassabov, and Alexander Lubotzky. Presentations of finite simple groups: a computational approach. *arXiv preprint arXiv:0804.1396*, 2008.

8. Martin Hofmann and Ulrich Schöpp. Pointer programs and undirected reachability. In *LICS*, pages 133–142, 2009.

9. Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, October 1988.

10. Lu, Zhang, Poon, and Cai. Simulating undirected st-connectivity algorithms on uniform jags and nnjags. In *Algo. and Comp.*, volume 3827 of *LNCS*. 2005.

11. Poon. Space bounds for graph connectivity problems on node-named jags and node-ordered jags. In *Found of Comp Sci, 1993. Proc., 34th Ann Sym on*, 1993.

12. Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.

13. R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Inf.*, 26(3):279–284, November 1988.