Efficient Labeling for Reachability in Directed Acyclic Graphs

Maciej Dulęba
Institute of Computer Science, University of Wrocław, Poland

Paweł Gawrychowski
Institute of Computer Science, University of Wrocław, Poland

Wojciech Janczewski
Institute of Computer Science, University of Wrocław, Poland

Abstract

We consider labeling nodes of a directed graph for reachability queries. A reachability labeling scheme for such a graph assigns a binary string, called a label, to each node. Then, given the labels of nodes u and v and no other information about the underlying graph, it should be possible to determine whether there exists a directed path from u to v. By a simple information theoretical argument and invoking the bound on the number of partial orders, in any scheme some labels need to consist of at least $n/4$ bits, where n is the number of nodes. On the other hand, it is not hard to design a scheme with labels consisting of $n/2 + \mathcal{O}(\log n)$ bits. In the classical centralised setting, where a single data structure is stored as a whole, Munro and Nicholson designed a structure for reachability queries consisting of $n^2/4 + o(n^2)$ bits (which is optimal, up to the lower order term). We extend their approach to obtain a scheme with labels consisting of $n/3 + o(n)$ bits.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases informative labeling scheme, reachability, DAG

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2020.27

1 Introduction

A labeling scheme assigns a binary string, called a label, to each node in a graph. Then, it should be possible to compute some function defined on subsets of nodes using only labels of the nodes in that subset, and no other information about the whole graph. Formally, a labeling scheme for a family of graphs consists of two parts, an encoder and a decoder. The encoder receives a graph from the specified family and outputs the label of each node in this graph. The label replaces the unique id of a node and allows the decoder to evaluate the desired function using only labels of the relevant nodes. Therefore, such labeling schemes are often called *informative* [25]. Another way of thinking about such a scheme is that we want to distribute the description of a graph among its individual nodes.

The most important characteristic of a scheme is its size, defined as the maximum length of a label assigned to any node. Additionally, it is desirable that the decoder is able to evaluate the function efficiently, ideally in constant time assuming random access to all the relevant labels. Finally, the encoder should work in polynomial time, and sometimes optimising its running time is yet another goal.

Arguably the most basic example of a function considered in this model is adjacency: the decoder needs to answer whether two nodes are neighbours in the graph, using only their labels. Such a labeling scheme is closely connected to the notion of an induced universal graph for a given family of graphs, where the induced universal graph needs to contain each graph from the family as an induced subgraph. More precisely, it is well known that a family of graphs has an adjacency labeling scheme of size k bits if and only if there is an induced universal graph for that family with at most 2^k nodes [17]. This also extends in a natural way to directed graphs. The question of the minimal size of induced universal graphs has been already studied by Moon [22] several decades ago. Recently, Alstrup, Kaplan, Thorup and Zwick [7] proved that it is possible to construct an adjacency labeling scheme for undirected graphs with size $n/2 + \mathcal{O}(1)$, which is optimal up to an additive constant. They also obtained similar tight results for directed graphs, tournaments, and bipartite graphs. The additive constant was later improved by Alon [1] in the context of induced universal graphs. Alstrup, Dahlgaard, and Knudsen [4] constructed an adjacency labeling scheme for trees with size $\log n + \mathcal{O}(1)$, which is again optimal up to an additive constant. Numerous other functions were considered, both in terms of upper and lower bounds: distance [11–13], connectivity [18, 20], sibling or ancestor relationship [3], nearest common ancestor in trees [6, 15], routing [27] and flow [18]. Often more restricted classes of graphs are analysed, most notably planar graphs [8, 9], bounded degree graphs [2] and sparse graphs [5, 14, 21]. See [26] for a recent survey.

Reachability in directed graphs. We focus on the general class of directed graphs. Alstrup et al. [7] considered adjacency queries in such graphs and designed a scheme of size $n + 3$, with the obvious lower bound being $n - 1$. The natural next step is to consider reachability queries, in which given the labels of u and v the decoder should answer if there is a directed path from u to v. It is not hard to see that, by identifying and collapsing the strongly connected components, it is enough to focus on directed acyclic graphs (DAGs). To extend a scheme for reachability in DAGs to a scheme for reachability in directed graphs, we simply append $\mathcal{O}(\log n)$ bits denoting the id of a node in its strongly connected component to the label for every node. Furthermore, we can assume that we are given the transitive closure of a DAG, in which reachability is equivalent to adjacency.

Posets. Reachability queries in a DAG naturally correspond to comparing elements in a partially ordered set (poset). Kleitman and Rothschild [19] proved the following result on the number of posets.

▶ **Theorem 1** ([19]). *Let $P(n)$ denote the number of posets on n elements. There exists a constant $C > 0$ such that*

$$2^{n^2/4} \le P(n) \le 2^{n^2/4 + C\, n^{3/2} \log n}.$$

This means that supporting reachability queries in a DAG requires storing at least $n^2/4$ bits, while the straightforward representation as an upper triangular matrix takes about $n^2/2$ bits. Munro and Nicholson [24] designed a succinct data structure consisting of only $n^2/4 + o(n^2)$ bits for this problem.

▶ **Theorem 2** ([24]). *For any poset on n elements, there exists a data structure consisting of $n^2/4 + \mathcal{O}(n^2 \log\log n/\log n)$ bits supporting precedence queries in constant time.*

The main idea in their approach is based on the so-called Zarankiewicz problem, which asks about a lower bound on the number of edges in a bipartite graph guaranteeing that there exists a balanced biclique ($K_{q,q}$) subgraph. Their construction first flattens the DAG to ensure that there are not too many layers, namely $\mathcal{O}(\log n)$. Then, they iteratively extract balanced bicliques with $q = \Theta(\log n/\log\log n)$ as long as sufficiently many edges remain. The structure of a biclique allows them to encode two possible edges with just a single bit instead of two. Finally, the remaining (not too many) edges are stored explicitly.

Our result. Our starting point is a simple reachability labeling scheme for directed graphs of size $n/2 + \mathcal{O}(\log n)$ stated in Theorem 7 (we remark that the underlying idea was already used by Moon [22]). Then we translate the method of Munro and Nicholson to obtain a more effective scheme of size $n/3 + o(n)$.

▶ **Theorem 3.** *There exists a reachability labeling scheme for directed graphs on n nodes of size $n/3 + o(n)$, with the decoder working in constant time.*

While we largely follow the approach of Munro and Nicholson, it needs to be carefully inspected and tweaked as to distribute the stored information among the nodes. The additional ingredient is an unbalanced adjacency labeling scheme for bipartite graphs. Finally, we explain how to adjust the presented scheme to achieve the *average* label size of $n/4 + o(n)$ at the expense of increasing the maximum label size to $n/2 + o(n)$. Other tradeoffs are also possible. We remark that an upper bound of $n/4$ on the average label size is optimal due to Theorem 1, as given a labeling scheme for a DAG and all pairs of labels, the decoder can reconstruct the entire corresponding poset. As an immediate corollary of Theorem 3, we obtain the following result.

▶ **Corollary 4.** *There exists an induced universal graph of size $2^{n/3+o(n)}$ for the family of transitively closed directed graphs on n nodes.*

Overview of our approach. The label of every node consists of two parts. The encoder for our scheme operates on a decomposition of the graph into antichains called layers, with no edges between the nodes in the same layer. First, the layers are created based on the longest-paths decomposition. Second, we ensure that there are only $\mathcal{O}(\log n)$ layers by removing not too many edges and merging some of the layers into super-layers. Information about the removed edges is distributed among the first parts of the labels, each of them consisting of $o(n)$ bits. Third, we run the following procedure that keeps removing edges from the current graph while maintaining its decomposition into layers. We consider the first two layers of the current graph and decompose their nodes into balanced biclique subgraphs and the remaining nodes. This is the key part of the construction that, roughly speaking, allows us to compress the graph. The nodes from the bicliques are removed from the graph, and information about their incident edges is carefully distributed among the second parts of the labels of both the removed and the remaining nodes. After having guaranteed that the subgraph corresponding to the remaining nodes of the first two layers is sufficiently sparse, we merge them into one layer and repeat the reasoning. While the idea of first flattening and then extracting bicliques is due to Munro and Nicholson [24], we need to inspect all the ingredients and carefully balance distributing the stored information among the labels. As a result, we end up with labels of length $n/3 + o(n)$, and with some care the decoder can be implemented to work in constant time.

2 Preliminaries

We consider labeling the nodes of a directed graph for reachability queries. A labeling scheme for a family of directed graphs on n nodes, denoted $\mathcal{G}_n$, consists of an encoder and a decoder. The encoder receives a graph $G = (V, E) \in \mathcal{G}_n$ and assigns a distinct binary string (called the label) $\ell_G(u)$ to each node $u \in V$. We will usually omit the subscript and denote the label of u simply by $\ell(u)$. The decoder, given $\ell(u)$ and $\ell(v)$ for some $u, v \in V$, should return if there is a directed path from u to v in G. We stress that the decoder is not aware of G and only knows that $\ell(u)$ and $\ell(v)$ are labels of two nodes from the same graph $G \in \mathcal{G}_n$. We are interested in minimising the maximum length of a label, that is $\max_{G\in\mathcal{G}_n} \max_{u\in V} |\ell(u)|$, called the size of the labeling scheme. We are also going to consider minimising the average length of a label, defined as $\max_{G \in \mathcal{G}_n} \sum_{u \in V} |\ell(u)|/n$. When analysing the decoding time, we assume the standard Word RAM model with words of length $\Theta(\log n)$. That is, both labels are given as arrays, with each entry storing $\Theta(\log n)$ consecutive bits of the label, and the decoder can access any of these entries in constant time. To make our scheme more relevant for possible applications, we insist that the decoder is uniform, that is, actually works for any value of n (otherwise the set of inputs is possibly very large but finite, and the decoding procedure could simply access a preprocessed table, which is clearly not practical).

$u \rightsquigarrow v$ denotes that there is a directed path (possibly with zero length) from u to v, and in such case we say that u can reach v, or that v is greater than u.

We focus on the class of directed acyclic graphs on n nodes, denoted $\mathcal{DAG}_n$. A labeling scheme for $\mathcal{G}_n$ can be obtained from our construction for $\mathcal{DAG}_n$ using the following Lemma.

▶ **Lemma 5.** *Assume that there is a reachability labeling scheme for $\mathcal{DAG}_n$ of size $f(n)$ and average size $g(n)$, with the decoder working in constant time. Then there is also a reachability labeling scheme for $\mathcal{G}_n$ of size $f(n) + \mathcal{O}(\log n)$ and average size $g(n) + \mathcal{O}(\log n)$, with the decoder working in constant time.*

Proof. We explain how to obtain a labeling of the given directed graph $G = (V, E)$ by constructing a DAG $G' = (V, E')$, using the assumed scheme to label its nodes, and prepending some extra information to the label of every node.

G' is constructed by identifying the strongly connected components (SCCs) of G. Let $\{v_1, v_2, \ldots, v_k\}$ be the nodes in the same SCC. We add edges (v_i, v_{i+1}), for every $i = 1, 2, \ldots, k-1$, to E'. Then, for every edge $(u, v) \in E$ such that u and v belong to different SCCs consisting of nodes $\{u_1, u_2, \ldots, u_k\}$ and $\{v_1, v_2, \ldots, v_r\}$, respectively, we add the edge (u_k, v_1) to E'. It is easy to verify that, for any u and v belonging to different SCCs, $u \rightsquigarrow v$ in G if and only if $u \rightsquigarrow v$ in G'. We run our encoder on G' to obtain the label $\ell_{G'}(u)$ for every $u \in V$. Then, to obtain $\ell_G(u)$ we simply prepend the identifier of SCC of u, consisting of $\mathcal{O}(\log n)$ bits. This allows the decoder to correctly check if $u \rightsquigarrow v$ in G by first checking if they both belong to the same SCC, and if not inspecting $\ell_{G'}(u)$ and $\ell_{G'}(v)$. ◀

In the remaining part of the paper, we assume that the input graph $G = (V, E)$ is acyclic, and $G_c = (V, E_c)$ denotes its transitive closure. By definition, $u \rightsquigarrow v$ in G if and only if $(u, v) \in E_c$. Even though the graph is directed, we will also say that such u and v are adjacent.

To make the decoder computationally efficient, we need the following theorem of Hagerup, Miltersen and Pagh [16]:

▶ **Theorem 6.** *For a given set $S \subseteq \{0, 1, ..., n-1\}$ there is a dictionary of size $\mathcal{O}(|S|)$, allowing to answer queries $x \in S$ in constant time and constructible in time $\mathcal{O}(|S| \log |S|)$, assuming word size $\Theta(\log n)$.*

3 Warm-up and bipartite graphs

We first present a very simple preliminary scheme of size $n/2 + \mathcal{O}(\log n)$. We note that the underlying idea was already implicit in the work of Moon [22].

▶ **Theorem 7.** *There exists a reachability labeling scheme for $\mathcal{DAG}_n$ of size $n/2 + \mathcal{O}(\log n)$, with the decoder working in constant time.*

Proof. Consider $G = (V, E) \in \mathcal{DAG}_n$ and fix an arbitrary topological numbering of its nodes $I(\cdot)$, starting from 0. For any $u, v \in V$, $I(u) < I(v)$ implies that there is no path from v to u in G. The encoder for every node $u \in V$ composes $\ell(u)$ out of an encoding of $I(u)$ consisting of $\log n$ bits and a $\lfloor n/2 \rfloor$-bit table $B_u[\cdot]$. For $j = \{0, 1, \ldots, \lfloor n/2 \rfloor - 1\}$, the encoder sets $B_u[j] = 1$ iff the nodes u and $I^{-1}((I(u) + j + 1) \bmod n)$ are comparable, that is, $u \rightsquigarrow I^{-1}((I(u) + j + 1) \bmod n)$ or $I^{-1}((I(u) + j + 1) \bmod n) \rightsquigarrow u$. The size of this labeling scheme is $n/2 + \mathcal{O}(\log n)$. As for the decoder, it first extracts $I(u)$ and $I(v)$ from $\ell(u)$ and $\ell(v)$. If $I(u) = I(v)$ then $u = v$, if $I(u) > I(v)$ then there is no path from u to v. We are left with the case $I(u) < I(v)$. If $I(v) - I(u) \leq \lfloor n/2 \rfloor$ then the bit $B_u[I(v) - I(u) - 1]$ determines whether v is reachable from u. Otherwise the bit $B_v[n + I(u) - I(v) - 1]$ gives us this information. ◀

A technical ingredient in our solution is an adjacency labeling scheme for undirected bipartite graphs, or equivalently reachability queries for directed graphs consisting of two layers, with the edges directed from the first layer to the second layer. The bounds from the following lemma can be also inferred from the spreading lemma used by Alstrup, Kaplan, Thorup and Zwick [7] by setting all ℓ_i to be equal. In the appendix we provide a direct proof that avoids their round-robin procedure and allows us to provide a detailed description of the decoder.

▶ **Theorem 8.** *Set a, b and consider a family $\mathcal{K}_{a,b}$ of bipartite graphs with two layers A, B with a and b nodes correspondingly. For any natural α, β satisfying $a\alpha + b\beta > ab$ there exists an adjacency labeling scheme of size $\alpha + \mathcal{O}(\log N)$ for nodes from A and size $\beta + \mathcal{O}(\log N)$ for nodes from B, where $N = \max\{\alpha, \beta, a, b\}$, with the decoder working in constant time.*

We remark that it is not difficult to see that such a scheme exists, by applying Hall's marriage theorem on the following auxiliary bipartite graph $G' = (A', B'; E')$. We set $A' = A \times B$, $B' = (A \times \{0, 1, \ldots, \alpha - 1\}) \cup (B \times \{0, 1, \ldots, \beta - 1\})$, and connect $(u_a, u_b) \in A'$ to every node of the form $(u_a, i), i \in \{0, 1, \ldots, \alpha - 1\}$ and $(u_b, j), i \in \{0, 1, \ldots, \beta - 1\}$, creating $\alpha + \beta$ edges in total for every node from A'. When $a\alpha + b\beta \geq ab$ holds, this graph can be verified to admit a perfect matching by Hall's marriage theorem. Such a perfect matching forms an injective function from the edges of the original graph to the bits of the labels of desired size. However, we do not want the decoder to store the perfect matching, or to compute it upon a query, so we need an explicit construction.

4 DAG flattening

We are given a transitively closed directed acyclic graph $G_c = (V, E_c)$. Let $d[v]$ be the length of the longest directed path ending in node v, and $U_i = \{v \in V : d[v] = i - 1\}$. $\mathcal{U} = (U_1, U_2, \ldots, U_t)$ is a partition of nodes of the graph into antichains called *layers*. Clearly, there are no edges between the nodes in the same U_i, and by enumerating the nodes of $U_1, U_2, \ldots, U_t$ in this order we obtain a topological sorting of G_c.

Instead of iteratively merging pairs of adjacent layers, as done by Munro and Nicholson [24], we directly describe which layers should be merged. Let γ be a parameter. We call a layer U_i *thick* if $|U_i| > n/\gamma$, and *thin* otherwise. We merge intervals of consecutive layers to create *super-layers*. Each thick layer forms its own separate super-layer, whereas consecutive thin layers are glued into a single super-layer, up to the point where its size exceeds n/γ or when a thick layer is encountered. By construction each super-layer has one of the following types:

type 1 single thick layer,
type 2 consecutive thin layers with $(n/\gamma, 2n/\gamma]$ nodes in total,
type 3 consecutive thin layers with $\leq n/\gamma$ nodes in total.

Furthermore, because each super-layer of type 3 is either followed by a type 1 super-layer, or is the very last super-layer, there are $\mathcal{O}(\gamma)$ super-layers.

After having generated the set of super-layers $\mathcal{S}$, we partition the edges as follows:

$$E_1 = \bigcup_{S \in \mathcal{S}} (S \times S) \cap E_c, \quad E_2 = E_c \setminus E_1.$$

We will show how to assign labels $\ell_1(\cdot)$ that allow checking if $(u,v) \in E_1$ given $\ell_1(u)$ and $\ell_1(v)$.

▶ **Lemma 9.** *There is an assignment of labels $\ell_1(\cdot)$ consisting of $\mathcal{O}(\log n + n/\gamma)$ bits that allows checking in constant time if $(u,v) \in E_1$, given $\ell_1(u)$ and $\ell_1(v)$.*

Proof. Let $I(\cdot)$ be the topological ordering of G_c obtained from $\mathcal{U} = (U_1, U_2, \ldots, U_t)$. We describe how to obtain $\ell_1(u)$ for $u \in S_i \in \mathcal{S}$. As each super-layer consists of consecutive layers, nodes of S_i create an interval $[\mathsf{beg}_i, \mathsf{end}_i)$ in the topological ordering. Let thick_i be a Boolean value denoting whether S_i is type 1 super-layer. $\ell_1(u)$ consists of numbers $I(u)$, i, beg_i, end_i and bit thick_i. If $\mathsf{thick}_i = 0$ the encoder appends a bit-table $C_u[\cdot]$ of length $\mathsf{end}_i - \mathsf{beg}_i + 1 \le 2n/\gamma + 1$ to $\ell_1(u)$, where $C_u[j] = 1$ iff $(u, I^{-1}(\mathrm{beg}_i + j)) \in E_1$. If $\mathsf{thick}_i = 1$, then there are no edges inside this super-layer and the encoder does not append anything to $\ell_1(u)$. Observe that each label consists of $\mathcal{O}(\log n + n/\gamma)$ bits, and the decoder is straightforward to implement in constant time. ◀

After removing E_1 from G_c, we obtain a new graph $G'_c = (V, E_2)$ consisting of only $\mathcal{O}(\gamma)$ layers, as each super-layer now becomes a layer (however, the decomposition into layers is now not based on considering the longest paths). We set $\gamma = \log n$, this makes the labels obtained from Lemma 9 consist of only $o(n)$ bits, while the new graph G'_c consists of $\mathcal{O}(\log n)$ layers. It is easy to see that G_c is still transitively closed.

5 Flat DAG labeling

We are now given a transitively closed directed graph $G'_c = (V, E_2)$ with $\mathcal{O}(\log n)$ layers $U_1, U_2, \ldots, U_t$. Our goal is to find an adjacency labeling scheme for such graphs. As in [24] we will find and remove bicliques in consecutive layers, with the main tool being the following theorem by Mubayi and Turan.

▶ **Theorem 10** ([23]). *There exists a constant $c_{\min}$, such that every undirected graph $G = (V, E)$ with $|V| \ge c_{\min}$ and $|E| \ge 8|V|^{3/2}$ contains a biclique $K_{q,q}$, where $q = \Theta(\log|V| / \log(|V|^2/|E|))$. This biclique can be found in $\mathcal{O}(|E|)$ time.*

We remark that the above theorem will be applied only on bipartite graphs that are much denser than the required threshold of $8|V|^{3/2}$. Additionally, any $q = \omega(1)$ would suffice for our approach. However, this does not seem to allow for a simpler proof.

5.1 Biclique decomposition

We are given an undirected bipartite graph $G_{\mathrm{bip}} = (A, B; E)$, and our goal in this subsection is to partition it into bicliques. Later, this procedure will be iteratively applied on two consecutive layers of the initial graph. Let $G_1 = G_{\mathrm{bip}}$. As long as the current graph satisfies the conditions of Theorem 10, we apply it to extract a biclique (removing its nodes and edges), and repeat. See Algorithm 1 for a detailed description of the procedure.

Algorithm 1 Decomposing an undirected bipartite graph into bicliques.

```
1: procedure FindBicliques($G_{\text{bip}} = (A, B; E)$)
2:     $G_1 \leftarrow G_{\text{bip}}$.
3:     $n \leftarrow |A| + |B|$
4:     $\mathcal{K} \leftarrow \emptyset$                                          ▷ family of bicliques
5:     $i \leftarrow 1$                                                  ▷ current graph index
6:     while $G_i = (A^i, B^i; E^i)$ satisfies $w = |A^i| + |B^i| > \max(c_{min}, n^{3/4})$, $|E^i| > w^2/(\log^6 w)$ do
   ▷ For large enough $n$, $|E^i| > w^2/(\log^6 w)$ implies $|E^i| > 8w^{3/2}$
7:         $K_i = (A_i, B_i; A_i \times B_i)$ is a biclique found by applying Theorem 10, $A_i \subseteq A^i$, $B_i \subseteq B^i$
8:         $\mathcal{K} \leftarrow \mathcal{K} \cup \{K_i\}$
9:         $G_{i+1} = (A^i \setminus A_i, B^i \setminus B_i; E^i \setminus (A_i \times B^i \cup A^i \times B_i))$ ▷ we remove $A_i$ and $B_i$ to obtain $G_{i+1}$
10:        $i \leftarrow i + 1$
   ▷ we obtain the final irreducible $G_i = (A^i, B^i; E^i)$
11:    $A_{\text{rest}} \leftarrow A^i$
12:    $B_{\text{rest}} \leftarrow B^i$
```

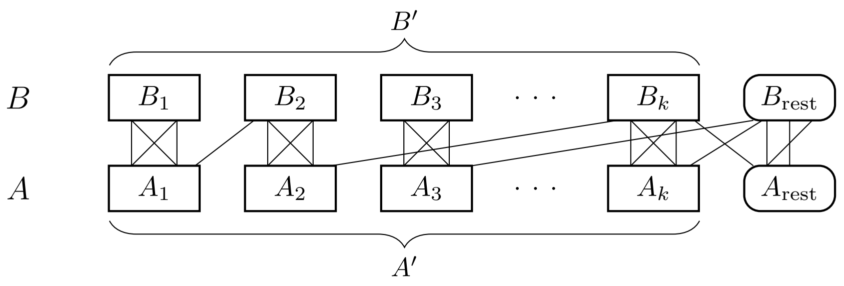

Figure 1 $G_{\text{bip}} = (A, B; E)$ partitioned into bicliques (A_i, B_i) and the leftovers $(A_{\text{rest}}, B_{\text{rest}})$.

Let k be the final iteration of Algorithm 1, and $A' = \bigcup_{i=1}^{k} A_i$, $B' = \bigcup_{i=1}^{k} B_i$. Clearly $A = A' \cup A_{\text{rest}}$ and $B = B' \cup B_{\text{rest}}$, see Figure 1. The obtained decomposition admits the following properties.

▶ **Lemma 11.** *Let $V_{\text{rest}} = A_{\text{rest}} \cup B_{\text{rest}}$ and $E_{\text{rest}} = E \cap (A_{\text{rest}} \times B_{\text{rest}})$. There exists a function $N_{\text{rest}} : V_{\text{rest}} \to \mathcal{P}(V_{\text{rest}})$ such that for every edge $(u, v) \in E_{\text{rest}}$ we have $u \in N_{\text{rest}}(v)$ or $v \in N_{\text{rest}}(u)$. Also $|N_{\text{rest}}(u)| = \mathcal{O}(n/\log^3 n)$ holds for every $u \in V_{\text{rest}}$.*

Proof. G_i does not satisfy condition from line 6 of Algorithm when A_{rest} and B_{rest} are created. There are two possible cases:

- $|A_{\text{rest}}| + |B_{\text{rest}}| = |A^i| + |B^i| \leq \max(c_{\min}, n^{3/4})$: set $N_{\text{rest}}(u) = \{v \in V_{\text{rest}} : (u, v) \in E_{\text{rest}}\}$, it satisfies all the conditions.
- $|E_{\text{rest}}| < w^2/(\log^6 w) \leq n^2/(\log^6 n)$: let $V_{\text{rest}} = V_{\text{big}} \cup V_{\text{small}}$, where V_{big} is a set of nodes having at least $n/\log^3 n$ incident edges in the set E_{rest}, whereas V_{small} are the remaining nodes. For $u \in V_{\text{small}}$ set $N_{\text{rest}}(u) = \{v \in V_{\text{rest}} : (u, v) \in E_{\text{rest}}\}$. For $u \in V_{\text{big}}$ set $N_{\text{rest}}(u) = \{v \in V_{\text{big}} : (u, v) \in E_{\text{rest}}\}$. In the second case $|N_{\text{rest}}(u)| \leq |V_{\text{big}}| \leq 2|E_{\text{rest}}|/(n/\log^3 n) \leq 2n/\log^3 n$. Finally, consider an edge $(u, v) \in E_{\text{rest}}$. If $u \in V_{\text{small}}$ then $v \in N_{\text{rest}}(u)$. Otherwise $u \in V_{\text{big}}$ and $u \in N_{\text{rest}}(v)$, no matter to which V_* set v belongs. ◀

▶ **Lemma 12.** *For every $1 \leq i \leq k$, $|A_i| = |B_i| = \Theta(\log n/\log\log n)$.*

Proof. We have $w = |A^i| + |B^i| > n^{3/4}$ and $|E^i| > w^2/(\log^6 w)$. Theorem 10 finds a biclique of size $\Theta(\log w/\log(w^2/|E^i|)) = \Theta(\log n/\log\log n)$. ◀

5.2 Encoding

We apply Algorithm 1 iteratively to decompose the whole $G'_c = (V, E_2)$. Let s be the number of the current iteration. We take the first two of the remaining layers, U_s and U_{s+1}, treat them as an undirected bipartite graph, and find its biclique decomposition using Algorithm 1. We obtain the set of bicliques $\mathcal{K}$, the leftovers $(A_{\text{rest}}, B_{\text{rest}})$, and the remaining layers $U_{s+2}, U_{s+3}, \ldots, U_t$. We will soon explain how to encode information about the edges E_{inter} connecting the nodes from bicliques to other nodes in the labels $\ell^s_{\text{inter}}(\cdot)$. We will also explain how to encode the information about the edges E_{in} between the nodes from (possibly different) bicliques and between A_{rest} and B_{rest} in the labels $\ell^s_{\text{in}}(\cdot)$. This allows us to remove all of these edges, and also all nodes from bicliques. We merge A_{rest}, B_{rest} to obtain a new layer replacing U_s and U_{s+1} and repeat the procedure. See Algorithm 2 for a detailed description, and Figure 2 for an illustration of a single iteration.

Algorithm 2 Decomposing consecutive layers of a flat DAG into bicliques.

```
1: procedure LabelFlatDAG($V, E_2, s; U_s, U_{s+1}, \ldots, U_t$)
2:     if $s = t$ then return
3:     $E_{s,s+1} \leftarrow E_2 \cap (U_s \times U_{s+1})$
4:     FindBicliques($U_s, U_{s+1}; E_{s,s+1}$) ▷ we obtain the set of bicliques $\mathcal{K}$ and leftovers $A_{\text{rest}}, B_{\text{rest}}$
5:     $V' \leftarrow A' \cup B'$, $V_{\text{rest}} \leftarrow A_{\text{rest}} \cup B_{\text{rest}}$, $\tilde{V} \leftarrow V \setminus (V' \cup V_{\text{rest}})$
6:     $E_{\text{inter}} \leftarrow E_2 \cap ((V' \times \tilde{V}) \cup (A' \times B_{\text{rest}}) \cup (A_{\text{rest}} \times B'))$
7:     $E_{\text{in}} \leftarrow E_2 \cap ((A' \times B') \cup (A_{\text{rest}} \times B_{\text{rest}}))$
8:     Store information about $E_{\text{in}}$ and $E_{\text{inter}}$ in the labels $\ell^s_{\text{in}}(\cdot)$, $\ell^s_{\text{inter}}(\cdot)$
9:     $E_2 \leftarrow E_2 \setminus (E_{\text{in}} \cup E_{\text{inter}})$          ▷ remove edges
10:    $V \leftarrow V \setminus V'$          ▷ remove nodes
11:    $U_{s+1} \leftarrow A_{\text{rest}} \cup B_{\text{rest}}$          ▷ create a new layer
12:    LabelFlatDAG($V, E_2, s+1; U_{s+1}, U_{s+2}, \ldots, U_t$).
```

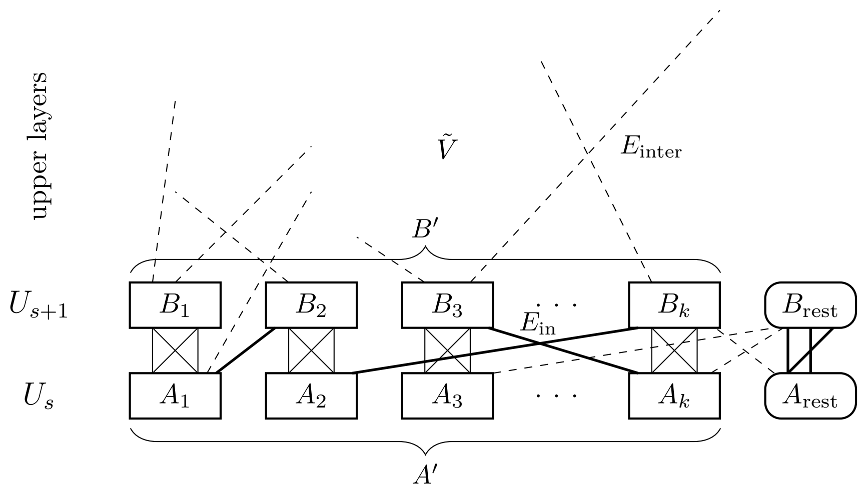

Figure 2 Layers U_s and U_{s+1} partitioned into bicliques (A_i, B_i) and the leftovers $(A_{\text{rest}}, B_{\text{rest}})$.

Irrespectively of the implementation of line 8, the number of iterations is $t = \mathcal{O}(\log n)$, and the current graph $G'_c = (V, E_2)$ remains transitively closed. We proceed to explain how to implement line 8. Fix an iteration s of the procedure. Let $V^s \subseteq V$ be the set of nodes and $E^s_2 \subseteq E_2$ the set of edges considered in this iteration, with V and E_2 referring to the initial graph G'_c. The auxiliary notation (A', B', A_{rest}, V' etc.) refers to the sets defined in the s-th iteration. We will also write just $\ell_{\text{in}}(\cdot)$ and $\ell_{\text{inter}}(\cdot)$ instead of $\ell^s_{\text{in}}(\cdot)$ and $\ell^s_{\text{inter}}(\cdot)$.

▶ **Lemma 13.** *There is an assignment of labels $\ell_{\text{in}}(\cdot)$ consisting of $|A'|/2 + \mathcal{O}(\log n)$ bits for the nodes of V', $\mathcal{O}(n/\log^2 n)$ bits for the nodes of V_{rest} and $\mathcal{O}(1)$ bits for the remaining nodes that allows checking in constant time if $(u, v) \in E_{\text{in}}$, given $\ell_{\text{in}}(u)$ and $\ell_{\text{in}}(v)$.*

Proof. For any $u \in V$, $\ell_{\text{in}}(u)$ consists of the following ingredients. First, we store an integer $\inf(u)$ encoding the information whether u was already removed from the graph, or which of the sets A', B', A_{rest}, B_{rest}, $\tilde{V}$ does it belong to. Then we have two cases:

$\boldsymbol{u \in V'}$: we append the label $\ell_{\text{bip}}(u)$ obtained by applying Theorem 8 on the bipartite graph $(A', B'; E_2^s \cap (A' \times B'))$ with parameters $\alpha = \beta = |A'|/2 + 1$,

$\boldsymbol{u \in V_{\text{rest}}}$: we append the structure described in Theorem 6 applied on the set $N_{\text{rest}}(u)$ from Lemma 11.

Given $\ell_{\text{in}}(u)$ and $\ell_{\text{in}}(v)$, we proceed as follows. By inspecting $\inf(u)$, $\inf(v)$ we can distinguish the following three options:

1. If $u \in A'$ and $v \in B'$, then using $\ell_{\text{bip}}(u)$ and $\ell_{\text{bip}}(v)$ we can check whether $(u, v) \in E_2 \cap (A' \times B')$.
2. If $u \in A_{\text{rest}}$ and $v \in B_{\text{rest}}$, then we can check whether $u \in N_{\text{rest}}(v)$ or $v \in N_{\text{rest}}(u)$ using the dictionaries stored in both labels.
3. Otherwise $(u, v) \notin E_{\text{in}}$.

It is straightforward to verify that the sizes of labels are as required and the check can be implemented in constant time. ◀

▶ **Lemma 14.** *Let $\alpha = \lceil |V^s|/3 - |A'|/2 \rceil$, $\beta = \lceil 2|A'|/3 \rceil$, and k be the number of bicliques found in the current iteration. There is an assignment of labels $\ell_{\text{inter}}(\cdot)$ consisting of $\alpha + \mathcal{O}(\log n)$ bits for the nodes of V', $\beta + \mathcal{O}(\log n) + k$ bits for the nodes of $V^s \setminus V'$ and $\mathcal{O}(1)$ bits for the remaining nodes, that allows checking in constant time if $(u, v) \in E_{\text{inter}}$, given $\ell_{\text{inter}}(u)$ and $\ell_{\text{inter}}(v)$.*

Proof. We first verify that

$$\alpha|A'| + \beta(|V^s| - 2|A'|) > |A'|(|V^s| - 2|A'|).$$

Now we construct an undirected bipartite graph $\hat{G} = (\hat{A}, \hat{B}; \hat{E})$. Every node of $\hat{B}$ corresponds to a node of $V^s \setminus V'$. The definition of $\hat{A}$ is more complicated. Recall that $A' = \bigcup_{i=1}^{k} A_i$ and $B' = \bigcup_{i=1}^{k} B_i$. The bicliques are balanced, so we have the natural pairing of the nodes in A_i and B_i. Therefore, we have a pairing of the nodes of A' and B'. Every node of $\hat{A}$ corresponds to such a pair of nodes $a \leftrightarrow b$, where $a \in A'$ and $b \in B'$. Thus $|\hat{A}| = |A'| = |B'|$.

Observe that if for some $y \in B_i$ and $z \in V^s$ we have $(y, z) \in E_{\text{inter}}$, then $(x, z) \in E_{\text{inter}}$ for every $x \in A_i$, by the graph being transitively closed and (A_i, B_i) being a biclique. Let $\hat{a} \in \hat{A}$ correspond to $a_j \leftrightarrow b_j$, where $a_j \in A_i$, $b_j \in B_i$ (we say that $\hat{a}$ corresponds to both a_j and b_j), and let $\hat{b} \in \hat{B}$ correspond to $b \in V^s \setminus V'$. Whether $(\hat{a}, \hat{b}) \in \hat{E}$ depends on the location of b in V^s and the edges in E_{inter}. Exactly one of the following cases occurs:

1. $b \in A_{\text{rest}}$, so b is not adjacent to A_i (in particular not to a_j): $(\hat{a}, \hat{b}) \in \hat{E}$ iff $(b_j, b) \in E_{\text{inter}}$,
2. $b \in B_{\text{rest}}$, so b is not adjacent to B_i (in particular not to b_j): $(\hat{a}, \hat{b}) \in \hat{E}$ iff $(a_j, b) \in E_{\text{inter}}$,
3. $b \in \tilde{V}$, b is adjacent to some node of B_i, so $(a_j, b) \in E_{\text{inter}}$: $(\hat{a}, \hat{b}) \in \hat{E}$ iff $(b_j, b) \in E_{\text{inter}}$,
4. $b \in \tilde{V}$, b is not adjacent to any node of B_i, so $(b_j, b) \notin E_{\text{inter}}$: $(\hat{a}, \hat{b}) \in \hat{E}$ iff $(a_j, b) \in E_{\text{inter}}$.

We apply Theorem 8 on $\hat{G}$ with parameters α, β to obtain the labels $\ell_{\text{bip}}(\cdot)$. For any $u \in V$, $\ell_{\text{inter}}(u)$ consists of the following ingredients. First, we store an integer $\inf(u)$ encoding the information whether u was already removed from the graph, or which of the sets A', B', A_{rest}, B_{rest}, $\tilde{V}$ does it belong to. Second, we append $\ell_{\text{bip}}(\hat{u})$, where $\hat{u}$ corresponds to u in $\hat{G}$. Then we have two cases:

$\boldsymbol{u \in V'}$: we append the index i such that $u \in A_i \cup B_i$,

$\boldsymbol{u \in V^s \setminus V'}$: we append a bit-table $\mathrm{B}_u[\cdot]$ of length k, in which $\mathrm{B}_u[i]$ stores the information whether u is adjacent to some node of B_i.

Given $\ell_{\text{inter}}(u)$ and $\ell_{\text{inter}}(v)$, we proceed as follows. First we verify that $u, v \in V^s$ using $\inf(u)$ and $\inf(v)$, as otherwise $(u,v) \notin E_{\text{inter}}$. Let $\hat{u}, \hat{v}$ correspond to u and v in $\hat{G}$. By inspecting $\inf(u)$ and $\inf(v)$, we can check if $\hat{u}$ and $\hat{v}$ belong to $\hat{A}$ or $\hat{B}$. If $\hat{u}$, $\hat{v}$ belong both to the $\hat{A}$ or $\hat{B}$, then $(u,v) \notin E_{\text{inter}}$ and we are done. By swapping u and v we can thus assume that $\hat{u} \in \hat{A}$ and $\hat{v} \in \hat{B}$. Using $\ell_{\text{bip}}(\hat{u})$ and $\ell_{\text{bip}}(\hat{v})$ we can then check if $(\hat{u}, \hat{v}) \in \hat{E}$. From $\ell_{\text{inter}}(u)$ we extract the index i such that $u \in A_i \cup B_i$, and by additionally inspecting $\inf(u)$ we know if $u \in A_i$ or $u \in B_i$. By inspecting $\inf(v)$ we know whether $v \in A_{\text{rest}}$, $v \in B_{\text{rest}}$, or $v \in \tilde{V}$, and by accessing the appropriate entry of $\mathrm{B}_v[\cdot]$ we know if v is adjacent to some node of B_i. This allows us to distinguish between the four possible cases and check if $(u,v) \in E_{\text{inter}}$. In more detail, we have the following possibilities:

1. $v \in A_{\text{rest}}$, if $u \in A_i$ then we return false, and if $u \in B_i$ we return $(\hat{u}, \hat{v}) \in \hat{E}$,
2. $v \in B_{\text{rest}}$, if $u \in B_i$ then we return false, and if $u \in A_i$ then we return $(\hat{u}, \hat{v}) \in \hat{E}$,
3. $v \in \tilde{V}$ and v is adjacent to some node of B_i, if $u \in A_i$ we return true, and if $u \in B_i$ we return $(\hat{u}, \hat{v}) \in \hat{E}$,
4. $v \in \tilde{V}$ and v is not adjacent to any node of B_i, if $u \in A_i$ we return $(\hat{u}, \hat{v}) \in \hat{E}$, and if $u \in B_i$ we return false.

It is straightforward to verify that the sizes of labels are as required and the check can be implemented in constant time. ◀

Note that in some sense the four cases from the proof of Lemma 14, by the structure of the found bicliques, allow us to store information about two possible edges $((a_j, b), (b_j, b))$ in just a single bit. In a similar way, Munro and Nicholson were able to obtain their centralised structure consisting of $n^2/4 + o(n^2)$ bits. Unfortunately, for a labeling scheme, when the existence of an edge from E_{inter} is remembered by a node from $\hat{A}$, one bit is used in the labels of both a_j and b_j. Still, only a single bit is used when the existence of an edge is stored by a node from $\hat{B}$. This allows us to achieve a nontrivial upper bound on the total length of the label.

▶ **Lemma 15.** *For every $u \in V$, $\sum_{s=1}^{t} |\ell_{\text{in}}^s(u)| + |\ell_{\text{inter}}^s(u)| = n/3 + o(n)$.*

Proof. Let i be the iteration in which u is removed from the graph. Recall that V^i is the set of nodes considered in the i-th iteration, and let A'_s denote set A' in the s-th iteration. By Lemma 13, the length of $\ell_{\text{in}}^s(u)$ is:

$$\begin{array}{rl} \mathcal{O}(n/\log^2 n) & \text{for} \quad s < i \\ |A'_i|/2 + \mathcal{O}(\log n) & \text{for} \quad s = i \\ \mathcal{O}(1) & \text{in other cases.} \end{array}$$

This overall sums up to $o(n) + |A'_i|/2$ bits, as $t = \mathcal{O}(\log n)$. By Lemma 14, the length of $\ell_{\text{inter}}^s(u)$ is:

$$\begin{array}{rl} \lceil 2|A'_s|/3 \rceil + \mathcal{O}(\log n) + k_s & \text{for} \quad s < i \\ \lceil |V^i|/3 - |A'_i|/2 \rceil + \mathcal{O}(\log n) & \text{for} \quad s = i \\ \mathcal{O}(1) & \text{in other cases,} \end{array}$$

where k_s is the number of found bicliques in the s-th iteration. The sum of $2|A'_s|$ over all iterations $s < i$ is equal to the number of removed nodes until the i-th iteration, which is $n - |V^i|$. The sum of k_s is not greater than the number of found bicliques. Because each biclique is of size $\Theta(\log n / \log\log n)$, this number is $o(n)$. This makes the whole sum:

$$o(n) + |A'_i|/2 + (n - |V^i|)/3 + o(n) + |V^i|/3 - |A'_i|/2 + \mathcal{O}(\log n) = n/3 + o(n). \qquad ◀$$

5.3 Decoding

We define the label $\ell_2(u)$ to be the concatenation of all the labels $\ell^s_{\text{in}}(u)$ and $\ell^s_{\text{inter}}(u)$ generated by Algorithm 2 for $s = 1, 2, \ldots, t$. Additionally, we store $\mathcal{O}(\log n)$ indices denoting where every $\ell^s_{\text{in}}(u)$ and $\ell^s_{\text{inter}}(u)$ begins and ends in $\ell_2(u)$. As each index needs $\mathcal{O}(\log n)$ bits, this takes $\mathcal{O}(\log^2 n)$ extra bits stored in the very beginning of the label, and allows us to access any $\ell^s_{\text{in}}(u)$ and $\ell^s_{\text{inter}}(u)$ in constant time. Additionally, $\ell_2(u)$ stores two numbers $\text{Del}(u)$ and $\text{IU}(u)$, each in $\mathcal{O}(\log\log n)$ bits. $\text{Del}(u)$ is the last iteration in which u is present in the graph, that is, the largest s such that $u \in V^s$. $\text{IU}(u)$ is the index of the initial layer of u in G'_c, that is, i such that $u \in U_i$. By Lemma 15, $|\ell_2(u)| = n/3 + o(n)$.

▶ **Lemma 16.** *Given $\ell_2(u)$ and $\ell_2(v)$ we can check in constant time if $(u, v) \in E_2$.*

Proof. Every edge in E_2 ends up in exactly one of the sets E_{in} or E_{inter} defined in some iteration. Note that we do not have enough time to consider all possible iterations. Thus, we will first calculate the relevant iteration s, and then use $\ell^s_{\text{in}}(u)$, $\ell^s_{\text{inter}}(u)$, $\ell^s_{\text{in}}(v)$ and $\ell^s_{\text{inter}}(v)$ to check if $(u, v) \in E_2$. We will make sure that s is the unique iteration such that one of the sets E_{in} or E_{inter} might contain (u, v).

Assume that $\text{IU}(u) \le \text{IU}(v)$, as otherwise from the topological ordering $(u, v) \notin E_2$. If $\text{IU}(v) \le 2$, we take $s = 1$ as the edges between the first two layers are considered only in the first iteration. If $\text{IU}(v) > 2$ then we have two cases:

Del$(u) <$ IU$(v) - 1$: u was removed in the $\text{Del}(u)$-th iteration, and before this iteration v is not in the first two layers, so we take $s = \text{Del}(u)$,

Del$(u) \ge$ IU$(v) - 1$: after the $(\text{IU}(v) - 1)$-th iteration both u and v are in the first layer (or not in the graph anymore) and u is not in any biclique before that iteration, so we take $s = \text{IU}(v) - 1$.

Having identified the appropriate s, we use $\ell^s_{\text{in}}(u)$, $\ell^s_{\text{in}}(v)$ to check if $(u, v) \in E_{\text{in}}$ and $\ell^s_{\text{inter}}(u)$, $\ell^s_{\text{inter}}(v)$ to check if $(u, v) \in E_{\text{inter}}$, where E_{in} and E_{inter} are defined in the s-th iteration, in constant time. ◀

6 Conclusions

Lemmas 9 and 16 allow us to formulate the final theorem.

▶ **Theorem 3.** *There exists a reachability labeling scheme for directed graphs on n nodes of size $n/3 + o(n)$, with the decoder working in constant time.*

Proof. By Lemma 5, it is enough to construct a reachability labeling scheme for directed acyclic graphs on n nodes of size $n/3 + o(n)$ and the decoder working in constant time. Let $G = (V, E)$ be such a DAG, and $G_c = (V, E_c)$ its transitive closure. First, we flatten G_c to obtain a new DAG $G'_c = (V, E_2)$ consisting of $\mathcal{O}(\log n)$ layers. The set of removed edges E_1 is encoded in the labels $\ell_1(\cdot)$ as described in Lemma 9, using $o(n)$ bits in the label of each node and allowing checking if $(u, v) \in E_1$ given the labels of u and v, in constant time. Next, we proceed as described in Section 5 to obtain the labels $\ell_2(\cdot)$. By Lemma 15, this uses $n/3+o(n)$ bits in the label of each node and by Lemma 16 allows checking if $(u,v) \in E_2$ given the labels of u and v in constant time. Finally, the label of each node u is the concatenation of $\ell_1(u)$ and $\ell_2(u)$, with appropriate padding as to make the length of both parts known and allow accessing any of them in constant time. ◀

We note that the scheme can be tweaked to guarantee the optimal (up to second-order term) *average* size $n/4$, matching the centralised bound.

▶ **Theorem 17.** *There exists a reachability labeling scheme for directed graphs on n nodes of* average *size $n/4+o(n)$, maximum size $n/2+o(n)$, and with the decoder working in constant time.*

Proof Sketch. To this end, we just modify Lemma 14, setting $\alpha = 0$ and $\beta = |A'|+1$. Then the whole set E_{inter} is remembered by the nodes in further layers, and no information about these edges is stored by the nodes from V'. The method from Lemma 13 stays intact, so the nodes from V_{rest} store $o(n)$ bits and the nodes from V' store $|A'|/2+\mathcal{O}(\log n)$ bits. After that change, take any node u and assume it is removed in the i-th iteration. Then, $\ell_2(u)$ uses one bit for every two nodes removed in the previous iterations and one bit for every four nodes removed in the i-th iteration. More precisely, recall that A'_s denotes the size of set A' in s-th iteration of the Algorithm 2, and let V_{prev} be the set of nodes erased from the graph before iteration i. Then, the label of u consists of the following elements:

- Label $\ell_1(u)$ from Lemma 9, which has length $o(n)$.
- Labels $\ell^s_{\text{in}}(u)$, with total size of $A'_i/2+o(n)$ bits as in the previous scheme.
- Labels $\ell^s_{\text{inter}}(u)$. They have lengths $|A'_s|+\mathcal{O}(\log n)+k_s$ for iterations $s<i$ and $\mathcal{O}(\log n)$ for the other iterations, so the sum of their sizes is $|V_{\text{prev}}|/2+o(n)$.
- Small additional information, that is indices denoting beginning of each sublabel and numbers $\text{Del}(u)$ and $\text{IU}(u)$.

Let us number the nodes in order of being erased from the graph, and say nodes from the A'_i erased in iteration i received numbers in $[a_i, b_i]$. Then length of the label for node u is $a_i/2+(b_i-a_i)/4+o(n)$. It is easy to verify that the sum of the lengths of all the labels is at most $n^2/4+o(n)$. This is paid for with unbalanced labels, as after the described change to $\ell^s_{\text{inter}}(\cdot)$ maximum size is bounded by $n/2+o(n)$ (with the nodes from further layers having longer labels than the nodes from the previous layers). ◀

By improving on the simple upper bound of $n/2+\mathcal{O}(\log n)$, our result brings us closer to resolving the natural question of the space complexity of reachability labeling for directed graphs. The only lower bound on the worst-case (and also average) size of a label in such a scheme is $n/4$, following from the result on the number of posets, and our scheme achieves an upper bound of $n/3+o(n)$. We remark that it does not seem possible to decrease the upper bound achieved by our scheme by simply tweaking the parameters, so new ideas are required.

References

1 Noga Alon. Asymptotically optimal induced universal graph. *Geometric and Functional Analysis*, 27(1):1–32, 2017.

2 Noga Alon and Rajko Nenadov. Optimal induced universal graphs for bounded-degree graphs. In *28th SODA*, pages 1149–1157, 2017.

3 Stephen Alstrup, Philip Bille, and Theis Rauhe. Labeling schemes for small distances in trees. *SIAM Journal on Discrete Mathematics*, 19(2):448–462, 2005.

4 Stephen Alstrup, Søren Dahlgaard, and Mathias Bæk Tejs Knudsen. Optimal induced universal graphs and adjacency labeling for trees. In *56th FOCS*, pages 1311–1326, 2015.

5 Stephen Alstrup, Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Ely Porat. Sublinear distance labeling. In *24th ESA*, pages 5:1–5:15, 2016.

6 Stephen Alstrup, Esben Bistrup Halvorsen, and Kasper Green Larsen. Near-optimal labeling schemes for nearest common ancestors. In *25th SODA*, pages 972–982, 2014.

7 Stephen Alstrup, Haim Kaplan, Mikkel Thorup, and Uri Zwick. Adjacency labeling schemes and induced-universal graphs. In *47th STOC*, pages 625–634. ACM, 2015.

8 Marthe Bonamy, Cyril Gavoille, and Michal Pilipczuk. Shorter labeling schemes for planar graphs. In *31st SODA*, pages 446–462, 2020.

9 Vida Dujmovic, Louis Esperet, Gwenaël Joret, Cyril Gavoille, Piotr Micek, and Pat Morin. Adjacency labelling for planar graphs (and beyond). *CoRR*, abs/2003.04280, 2020. `arXiv:2003.04280`.

10 Tomás Feder and Rajeev Motwani. Clique partitions, graph compression and speeding-up algorithms. In *23rd STOC*, pages 123–133, 1991.

11 Ofer Freedman, Paweł Gawrychowski, Patrick K. Nicholson, and Oren Weimann. Optimal distance labeling schemes for trees. In *36th PODC*, pages 185–194, 2017.

12 Cyril Gavoille, Michal Katz, Nir A. Katz, Christophe Paul, and David Peleg. Approximate distance labeling schemes. In *9th ESA*, pages 476–487, 2001.

13 Cyril Gavoille, David Peleg, Stéphane Pérennès, and Ran Raz. Distance labeling in graphs. In *12th SODA*, pages 210–219, 2001.

14 Paweł Gawrychowski, Adrian Kosowski, and Przemysław Uznański. Sublinear-space distance labeling using hubs. In *30th DISC*, pages 230–242, 2016.

15 Paweł Gawrychowski, Fabian Kuhn, Jakub Łopuszański, Konstantinos Panagiotou, and Pascal Su. Labeling schemes for nearest common ancestors through minor-universal trees. In *29th SODA*, pages 2604–2619, 2018.

16 Torben Hagerup, Peter Bro Miltersen, and Rasmus Pagh. Deterministic dictionaries. *Journal of Algorithms*, 41(1):69–85, 2001.

17 Sampath Kannan, Moni Naor, and Steven Rudich. Implicit representation of graphs. *SIAM Journal on Discrete Mathematics*, 5(4):596–603, 1992.

18 Michal Katz, Nir A. Katz, Amos Korman, and David Peleg. Labeling schemes for flow and connectivity. *SIAM J. Comput.*, 34(1):23–40, 2004.

19 D.J. Kleitman and B. L. Rothschild. The number of finite topologies. *Proceedings of the American Mathematical Society*, 25:276–282, 1970.

20 Amos Korman. Labeling schemes for vertex connectivity. *ACM Trans. Algorithms*, 6(2):39:1–39:10, 2010.

21 Adrian Kosowski, Przemysław Uznański, and Laurent Viennot. Hardness of exact distance queries in sparse graphs through hub labeling. In *38th PODC*, pages 272–279, 2019.

22 J. W. Moon. On minimal n-universal graphs. *Proceedings of the Glasgow Mathematical Association*, 7(1):32–33, 1965.

23 D. Mubay and G. Turan. Finding bipartite subgraphs efficiently. *Information Processing Letters*, 110(5):174–177, 2010.

24 J. Ian Munro and Patrick K. Nicholson. Succinct posets. *Algorithmica*, 76(2):445–473, 2016.

25 David Peleg. Informative labeling schemes for graphs. *Theor. Comput. Sci.*, 340(3):577–593, 2005.

26 Noy Galil Rotbart. *New Ideas on Labeling Schemes.* PhD thesis, University of Copenhagen, 2016.

27 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *13th SPAA*, pages 1–10, 2001.

A Labels for bipartite graphs in constant time

▶ **Theorem 8.** *Set a, b and consider a family $\mathcal{K}_{a,b}$ of bipartite graphs with two layers A, B with a and b nodes correspondingly. For any natural α, β satisfying $a\alpha + b\beta > ab$ there exists an adjacency labeling scheme of size $\alpha + \mathcal{O}(\log N)$ for nodes from A and size $\beta + \mathcal{O}(\log N)$ for nodes from B, where $N = \max\{\alpha, \beta, a, b\}$, with the decoder working in constant time.*

Proof. Given a graph $G = (A, B; E) \in \mathcal{K}_{a,b}$, the encoder E_{bip} first assigns numbers $\{0, 1, \dots, a-1\}$ to the nodes of A and numbers $\{a, a+1, \dots, a+b-1\}$ to the nodes from B. Call this assignment $I : A \cup B \to \mathcal{N}$. From now on we identify the nodes with their numbers. The label $\ell_{\text{bip}}(u)$ of a node u consists of the assigned number $I(u)$, parameters a, b, α, β ($\mathcal{O}(\log N)$ bits in total) and a bit table $T_u[\cdot]$. If $u \in A$, the encoder sets

$$T_u[i] = 1 \iff (u, a + (\lceil bu/a \rceil + i) \bmod b) \in E, \text{ for } i = \{0, 1, \dots, \alpha - 1\}.$$

If $u \in B$, the encoder sets

$$T_u[j] = 1 \iff ((\lceil a(u-a)/b \rceil + j) \bmod a, u) \in E, \text{ for } j = \{0, 1, \dots, \beta - 1\}.$$

In total labels have size $\alpha + \mathcal{O}(\log N)$ for nodes from A and $\beta + \mathcal{O}(\log N)$ for nodes from B. Now we describe the decoder. Let u, $v \in A \cup B$. Using $\ell_{\text{bip}}(u)$ and $\ell_{\text{bip}}(v)$, the decoder has to determine whether $(u, v) \in E$. First, it can check whether both nodes belong to the same layer (based on $I(u)$, $I(v)$, and value a). Assume that the nodes are in different layers (otherwise they are not adjacent) and $u \in A, v \in B$ (by swapping the nodes if necessary). Let $i_a = I(u)$, $i_b = I(v) - a$. We have $i_a \in \{0, 1, \dots, a-1\}$, $i_b \in \{0, 1, \dots, b-1\}$. Let

$$i = (i_b - \lceil bi_a/a \rceil) \bmod b, \quad j = (i_a - \lceil ai_b/b \rceil) \bmod a.$$

If $i \in \{0, 1, \dots, \alpha - 1\}$, then $T_u[i] = 1 \iff (u, v) \in E$. If $j \in \{0, 1, \dots, \beta - 1\}$, then $T_v[j] = 1 \iff (u, v) \in E$. In both cases, the decoder can look at the right bit of the table and answer the question $(u, v) \in E$ in constant time. So it is enough to show that for every i_a, i_b at least one of the above holds. When $\alpha \geq b$ or $\beta \geq a$ thesis is trivially satisfied for all i_a, i_b. Otherwise

$$i = \left(i_b - \left\lceil \frac{bi_a}{a} \right\rceil\right) \bmod b = \left\lfloor \frac{ai_b - bi_a}{a} \right\rfloor \bmod b,$$

$$j = \left(i_a - \left\lceil \frac{ai_b}{b} \right\rceil\right) \bmod a = \left\lfloor \frac{bi_a - ai_b}{b} \right\rfloor \bmod a.$$

Let $w = ai_b - bi_a$. From the constraints on i_a, i_b:

$$-b(a-1) \leq w \leq a(b-1).$$

If $w = 0$, then $i = j = 0$ and we are done. Suppose that $w > 0$, the opposite case is similar. We have

$$i < \alpha \iff \left\lfloor \frac{w}{a} \right\rfloor \bmod b < \alpha \iff \left\lfloor \frac{w}{a} \right\rfloor < \alpha \iff w < a\alpha,$$

and

$$j < \beta \iff \left\lfloor \frac{-w}{b} \right\rfloor \bmod a < \beta \iff \left\lfloor \frac{ab - w}{b} \right\rfloor < \beta$$
$$\iff ab - w < b\beta \iff ab - b\beta < w.$$

From the assumption $ab - b\beta < a\alpha$, thus at least one of the above inequalities is satisfied. ◀