Exact and Approximate Digraph Bandwidth

Pallavi Jain
Ben-Gurion University of the Negev, Beer-Sheva, Israel, pallavijain.t.cms@gmail.com

Lawqueen Kanesh
The Institute of Mathematical Sciences, HBNI, India, lawqueen@imsc.res.in

William Lochet
University of Bergen, Norway, William.Lochet@uib.no

Saket Saurabh
The Institute of Mathematical Sciences, HBNI, India, saket@imsc.res.in

Roohani Sharma
The Institute of Mathematical Sciences, HBNI, India, roohani@imsc.res.in

Abstract

In this paper, we introduce a directed variant of the classical Bandwidth problem and study it from the view-point of moderately exponential time algorithms, both exactly and approximately. Motivated by the definitions of the directed variants of the classical Cutwidth and Pathwidth problems, we define Digraph Bandwidth as follows. Given a digraph D and an ordering σ of its vertices, the *digraph bandwidth* of σ with respect to D is equal to the maximum value of $\sigma(v)-\sigma(u)$ over all arcs (u,v) of D going forward along σ (that is, when $\sigma(u) < \sigma(v)$). The Digraph Bandwidth problem takes as input a digraph D and asks to output an ordering with the minimum digraph bandwidth. The undirected Bandwidth easily reduces to Digraph Bandwidth and thus, it immediately implies that Directed Bandwidth is NP-hard. While an $\mathcal{O}^\star(n!)$[1] time algorithm for the problem is trivial, the goal of this paper is to design algorithms for Digraph Bandwidth which have running times of the form $2^{\mathcal{O}(n)}$. In particular, we obtain the following results. Here, n and m denote the number of vertices and arcs of the input digraph D, respectively.

- Digraph Bandwidth can be solved in $\mathcal{O}^\star(3^n \cdot 2^m)$ time. This result implies a $2^{\mathcal{O}(n)}$ time algorithm on sparse graphs, such as graphs of bounded average degree.
- Let G be the underlying undirected graph of the input digraph. If the treewidth of G is at most t, then Digraph Bandwidth can be solved in time $\mathcal{O}^\star(2^{n+(t+2)\log n})$. This result implies a $2^{n+\mathcal{O}(\sqrt{n}\log n)}$ algorithm for directed planar graphs and, in general, for the class of digraphs whose underlying undirected graph excludes some fixed graph H as a minor.
- Digraph Bandwidth can be solved in $\min\{\mathcal{O}^*(4^n \cdot b^n), \mathcal{O}^*(4^n \cdot 2^{b\log b\log n})\}$ time, where b denotes the optimal digraph bandwidth of D. This allow us to deduce a $2^{\mathcal{O}(n)}$ algorithm in many cases, for example when $b \leq \frac{n}{\log^2 n}$.
- Finally, we give a *(Single) Exponential Time Approximation Scheme* for Digraph Bandwidth. In particular, we show that for any fixed real $\epsilon > 0$, we can find an ordering whose digraph bandwidth is at most $(1+\epsilon)$ times the optimal digraph bandwidth, in time $\mathcal{O}^*(4^n \cdot (\lceil 4/\epsilon \rceil)^n)$.

2012 ACM Subject Classification Parameterized complexity and exact algorithms; Theory of computation → Approximation algorithms analysis

Keywords and phrases directed bandwidth, approximation scheme, exact exponential algorithms

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

[1] The $\mathcal{O}^\star$ notation hides the polynomial factors in the instance size.

© Pallavi Jain, Lawqueen Kanesh, Saket Saurabh, Roohani Sharma, Lochet William;
licensed under Creative Commons License CC-BY
42nd Conference on Very Important Topics (CVIT 2016).
Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:26

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The Bandwidth problem is a famous combinatorial problem, where given an undirected graph G on n vertices, the goal is to embed its vertices onto an integer line such that the maximum stretch of any edge of G is minimized. More formally, given a graph G on n vertices and an ordering $\sigma : V(G) \to [n]$, the *bandwidth of σ with respect to G* is $\max_{(u,v)\in E(G)}\{|\sigma(u) - \sigma(v)|\}$. In the Bandwidth problem, given a graph G, the goal is to find an ordering $\sigma : V(G) \to [n]$, which has *minimum bandwidth* with respect to G. The bandwidth problem has found applications in an array of fields including, but not limited to, the design of faster matrix operations computation on sparse matrices, VLSI circuit design, reducing the search space of constraint satisfaction problems and problems from molecular biology [22]. In many of the real world applications, a fundamental principle that the Bandwidth problem captures is that of delays that occur as a result of allocation of tasks on the time interval that have dependencies among them. An ordering in many scenarios represent the allocation of tasks/objects on a time-line/one-dimensional hardware, and the stretch of an edge captures the delay/effort/expense incurred to reach the other end of the edge.

One restriction on the kind of models captured by Bandwidth is that, the models cannot be tuned to allow for asymmetry or bias. More specifically, what happens when the connections available between the tasks/objects are unidirectional? What happens when there is a bias in terms of delay/expense based on the direction of communication on the time-line/one-dimensional hardware? The above inquisitivities lead to our first contribution to this article, which is the concept of Digraph Bandwidth[2]. Given a directed graph D on n vertices and an ordering $\sigma : V(D) \to [n]$, the *digraph bandwidth* of σ with respect to D is the *maximum stretch of the forward arcs in the ordering*, that is, $\max_{\substack{(u,v)\in E(D)\\ \sigma(u)<\sigma(v)}}\{\sigma(v) - \sigma(u)\}$. The Digraph Bandwidth problem takes as input a digraph D and outputs an ordering $\sigma : V(D) \to [n]$ with the least possible *digraph bandwidth* with respect to D.

Observe that, with the introduction of directions in the input graph, Digraph Bandwidth allows us to capture one-way dependencies, that can help in modelling scenarios where the links available for modelling the communication are one-directional. Also, by allowing to care only about the stretch of the forward arcs in the ordering, one can model channels where communication in one direction is cheaper/easier than the other. The later scenarios can occur while modelling an uphill-downhill communication, where the cost of going up is a matter of real concern whereas, the cost of going down is almost negligible. For a more explicit scenario which can be modelled by Digraph Bandwidth, but not necessarily by Bandwidth, refer to Appendix A.

Note that Digraph Bandwidth is indeed a generalization of the notion of undirected bandwidth, as for any graph G, if $\overleftrightarrow{G}$ denotes the digraph obtained from G by replacing each edge of G by one arc in both direction, then the bandwidth of G is equal to the directed bandwidth of $\overleftrightarrow{G}$. We would like to remark here that on the theoretical front, the way we lift the definition of bandwidth in undirected graphs to directed graphs, by considering the stretches of *only the forward arcs*, is not something unique that we do for Bandwidth. The idea of only considering arcs going in *one direction* for "optimizing some function" is very common to the directed setting. The simplest such example is the notion of a directed cut. If D is a digraph and X, Y are two disjoint subsets of vertices of D, then the *directed cut* of X and

[2] We choose the name Digraph Bandwidth over the more conventional Directed Bandwidth to avoid clash of names from literature (which will be discussed later).

Y, $\texttt{dcut}(X,Y)$, is defined as the set of arcs (u,v) in $E(D)$, where $u \in X$ and $v \in Y$. Another closely related notion is the notion of Directed Cutwidth introduced by Chudnosky *et al.* [5]. A digraph D on n vertices has cutwidth at most k if there exists an ordering of the vertices σ such that for every $i \in [n-1]$, $\texttt{dcut}(\{\sigma(1),\ldots,\sigma(i)\},\{\sigma(i+1),\ldots,\sigma(n)\})$ is at most k. Note that our notion of directed bandwidth is a stronger notion than cutwidth, as for any ordering σ, the cutwidth associated to σ is at most the digraph bandwith of σ. There is also a similar notion of Directed Pathwidth [5]. Observe that similar to Directed Cutwidth and Directed Pathwidth, Digraph Bandwidth is 0 on directed acyclic graphs (dags).

We would like to remark that ours is not the first attempt in generalising the definition of bandwidth for digraphs. A notion of bandwidth for directed graphs appeared in 1978 in the paper by Garey et al. [16]. But the notion was defined only for dags. In their problem, which they call Directed Bandwidth (DAG-BW), given a dag D, one is interested in finding a *topological* ordering (a linear ordering of vertices such that for every directed arc (u,v) from vertex u to vertex v, u comes before v in the ordering) of minimum bandwidth. Note that this is very different from our notion of Digraph Bandwidth which is always 0 for dags.

Algorithmic Perspective

Bandwidth is one of the most well-known and extensively studied graph layout problems [17]. The Bandwidth problem is NP-hard [25] and remains NP-hard even on very restricted subclasses of trees, like caterpillars of hair length at most 3 [24]. Furthermore, the bandwidth of a graph is NP-hard to approximate within a constant factor for trees [3]. Polynomial-time algorithms for the exact computation of bandwidth are known for a few graph classes including caterpillars with hair length at most 2 [2], cographs [29], interval graphs [20] and bipartite permutation graphs [19]. A classical algorithm by Saxe [26] solves Bandwidth in time $2^{\mathcal{O}(k)}n^{k+1}$, which is polynomial when k is a constant. In the realm of parameterized complexity, Bandwidth is known to be W[t]-hard for all $t \geq 1$, when parameterized by the bandwidth k of the input graph [4]. However, on trees it admits a parameterized approximation algorithm [12] and an algorithm with running time $2^{\mathcal{O}(k \log k)}n^{\mathcal{O}(1)}$ on AT-free graphs [18]. Unger showed in [27] that the problem is APX-hard. The best known approximation algorithm for this problem is due to Krauthgamer *et al.* [21] and it provides an $\mathcal{O}(\log^3 n)$ factor approximation.

The Bandwidth problem is one of the test-bed problems in the area of moderately exponential time algorithms and has been studied intensively. Trying all possible permutations of the vertex set yields a simple $\mathcal{O}^*(n!)$ time algorithm while the known algorithms for the problem with running time $2^{\mathcal{O}(n)}$ are far from straightforward. The $\mathcal{O}^*(n!)$ barrier was broken by Feige and Kilian [13] who gave an algorithm with running time $\mathcal{O}^\star(10^n)$. This result was subsequently improved by Cygan and Pilipczuk [6] down to $\mathcal{O}^\star(5^n)$. After a series of improvements, the current fastest known algorithm, due to Cygan and Pilipczuk [9, 7] runs in time $\mathcal{O}^\star(4.383^n)$. We also refer the readers to [8] for the best known exact algorithm running in polynomial space. For graphs of treewidth t, one can design an algorithm with running time $2^n n^{\mathcal{O}(t)}$ [1, 7]. On the other hand, Feige and Talwar [14] showed that the bandwidth of a graph of treewidth at most t can be $(1+\varepsilon)$-approximated in time $2^{\mathcal{O}(\log n(t+\sqrt{\frac{n}{\varepsilon}}))}$. Vassilevska *et al.* [28] gave a hybrid algorithm which after a polynomial time test, either computes the bandwidth of a graph in time $4^{n+o(n)}$, or provides $\gamma(n)\log^2 n \log\log n$-approximation in polynomial time for any unbounded γ. Moreover, for any two positive integers $k \geq 2$, $r \geq 1$, Cygan and Pilipczuk presented a $(2kr-1)$-approximation algorithm that solves Bandwidth for an arbitrary input graph in $\mathcal{O}(k^{\frac{n}{(k-1)r}} n^{\mathcal{O}(1)})$ time and polynomial space [7]. Finally, Fürer

et al. [15] gave a factor 2-approximate algorithm for BANDWIDTH running in time $\mathcal{O}(1.9797^n)$. DAG-BW, as defined by Garey *et al.* [16] for dags, was shown to admit a polynomial time algorithm for testing if a dag has bandwidth at most 2. Also, it was proved that the problem to determine if the directed bandwidth of a dag is at most k, for any $k > 2$, is NP-hard even in the case of oriented trees. This notion of directed bandwidth *reappeared* in [23], where it was studied for dense digraphs.

Our Results

The main objective of this paper is to *introduce a directed* variant of the BANDWIDTH problem for general digraphs and study it from the view point of moderately exponential time algorithms, both *exactly and approximately.* Throughout the remaining, n, m denote the number of vertices and arcs in the input digraph, respectively. For many linear layout problems on graphs on n vertices, beating even the trivial $\mathcal{O}^\star(n!)$ algorithm asymptotically remains a challenge. In this article we design $2^{\mathcal{O}(n)}$ time algorithms for DIGRAPH BANDWIDTH. Below we mention the challenges that DIGRAPH BANDWIDTH imposes when we try to apply the techniques used in the design of $2^{\mathcal{O}(n)}$ algorithm for BANDWIDTH, and how we bend our ways to overcome them to design the desired algorithms.

The $2^{\mathcal{O}(n)}$ time algorithms for BANDWIDTH that exist in literature (cited above), all follow a common principle of *bucket-then-order*. Suppose one is interested in checking whether the input graph has an ordering of bandwidth b. The *bucket-then-order* procedure is a 2-step procedure, where in the first step, instead of directly guessing the position of the vertex in the ordering, for a range of consecutive positions (called *buckets*) of size $\mathcal{O}(b)$, one guesses the set of vertices that will occupy these positions in the final ordering. This process of allocating a set of vertices to a range of consecutive positions is called *bucketing.* Since one can always assume that the graph is connected, once a bucket for the first vertex is guessed using n trials, its neighbours only have a choice of some c buckets for a small constant c depending on the constant in the order notation of the size of the bucket. This, makes the bucketing step run in time $2^{\mathcal{O}(n)}$. The outcome of the first step is a collection of bucketings which contains a bucketing that is "consistent" with the final ordering. In the second step, given such a consistent bucketing, one can find the final ordering using either a recursive divide and conquer technique or a dynamic programming procedure or a measure and conquer kind of an analysis.

In the case of DIGRAPH BANDWIDTH, finding a bucketing that is consistent with the final ordering becomes a challenge as even the *information that a vertex is placed in some fixed bucket does not decrease the options of the number of buckets in which its neighbours can be placed.* This is because there could be some out-neighbours (resp. in-neighbours) of it that need to be placed before (resp. after) it thereby contributing to backward arcs, which eventually results in the need for allocating them to far off buckets. We cope up with this challenge of bucketing in two ways - both of which lead to interesting algorithms that run in $2^{\mathcal{O}(n)}$ time in different cases. As a first measure of coping up, we take the strategy of "kill what cause you trouble". Formally speaking, it is the set of backward arcs in the final ordering that have unbounded stretch and hence, make the bucketting process difficult. One way to get back to the easy bucketting case is to guess the set of arcs that will appear as backward arcs in the final ordering. Having guessed these arcs, one can remove them from the graph and preserve the information that the arcs which remain all go forward the final ordering. This problem becomes similar to the DAG-BW problem defined on dags by Garey et al. [16]. We show that one can do the bucketing tricks similar to the undirected case here to design a $2^{\mathcal{O}(n)}$ algorithm for this problem (Theorem 1.1). This together with the initial

179 guessing of the backward arcs gives Theorem 1.2.

180 ▶ **Theorem 1.1.** DAG-BW *on dags can be solved in* $\mathcal{O}^\star(3^n)$ *time.*

181 ▶ **Theorem 1.2.** Digraph Bandwidth *can be solved in* $\mathcal{O}^\star(3^n \cdot 2^m)$ *time.*

182 Note that even though the 2^m in the running time of Theorem 1.2 looks expensive, it
183 already generates an algorithm better that $\mathcal{O}^\star(n!)$ for any digraph that has at most $o(n \log n)$
184 arcs. In particular, this implies an exact algorithm with running time $2^{\mathcal{O}(n)}$ whenever
185 $|E(D)| = \mathcal{O}(|V(D)|)$, for example for digraphs with bounded average degree.
186 We will now briefly explain about our second way of dealing with the bucketing phase.
187 As discussed earlier, getting a hold over the arcs which will go backward in the final ordering,
188 eases out the remaining process. In this strategy, instead of guessing the arcs that goes
189 backward by a brute force way (that takes 2^m), we exploit the fact that guessing a partition
190 of the vertex set into two parts, left and right - which corresponds to the first $n/2$ vertices in
191 the final ordering and the last $n/2$ vertices in the final ordering, also gives hold on *some* if not
192 all backward arcs in the final ordering. We place this simple observation into the framework
193 of a divide and conquer algorithm to get a bucketting that is not necessarily "consistent"
194 with the final ordering, but is not too far away to yield a "close enough" approximation to
195 the optimal ordering. This result is formalized in Theorem 1.3. Effectively, the result states
196 that one can find an ordering whose digraph bandwidth is at most $(1 + \epsilon)$ times the optimal
197 in time $\mathcal{O}^\star(1/\epsilon)^n$. Note that, this result is in contrast with the result of Feige and Talwar [14]
198 for undirected bandwidth where they gave an exponential time approximation scheme that
199 run in time which had a dependence on the treewidth of the graph($2^{\mathcal{O}(\log n(t+\sqrt{\frac{n}{\epsilon}}))}$). As a
200 side result of our strategy, we can also design an algorithm for solving Digraph Bandwidth
201 optimally on general digraphs in time $\mathcal{O}^\star(2^{\mathcal{O}(n)} \cdot OPT^n)$ or $\mathcal{O}^\star(2^{\mathcal{O}(n)} \cdot 2^{OPT \log OPT \log n})$,
202 where OPT is the optimal digraph bandwidth of the input digraph. This result is stated in
203 Theorem 1.4. Note that, on one hand where $\mathcal{O}(OPT^n)$ is easy to get for the undirected case
204 (because fixing the position of one vertex in the ordering leaves only $2 \cdot OPT$ choices for its
205 neighbours), it is not trivial for the directed case. Also, observe that Theorem 1.4 gives a
206 $2^{\mathcal{O}(n)}$ algorithm whenever $b \leq n/\log^2 n$.

207 ▶ **Theorem 1.3** ((Single) Exponential Time Approximation Scheme)**.** *For any real number*
208 $\epsilon > 0$*, for any digraph* D*, one can find an ordering of digraph bandwidth at most* $(1 + \epsilon)$
209 *times the optimal, in time* $\mathcal{O}^*(4^n \cdot (\lceil 4/\epsilon \rceil)^n)$*.*

210 ▶ **Theorem 1.4.** Digraph Bandwidth *can be solved in* $\min\{\mathcal{O}^*(4^n \cdot b^n), \mathcal{O}^*(4^n \cdot 2^{b \log b \log n})\}$
211 *time, where b is the optimal digraph bandwidth of the input digraph.*

212 Our last result is based on the connection of the Bandwidth problem with a subgraph
213 isomorphism problem. Amini *et al.* [1] viewed the Bandwidth problem, on undirected
214 graphs, as a subgraph isomorphism problem, and using an inclusion-exclusion formula with
215 the techniques of counting homomorphisms on graphs of bounded treewidth, they showed
216 that an optimal bandwidth ordering of a graph on n vertices of treewidth at most t can be
217 computed in time $\mathcal{O}^\star(2^{t \log n + n})$ and space $\mathcal{O}^\star(2^{t \log n})$. Using this approach and by relating
218 Digraph Bandwidth via directed homomorphisms to directed path-like-structures, we
219 obtain the following result.

220 ▶ **Theorem 1.5** ([3])**.** *Let* D *be a digraph on* n *vertices and* D' *be the underlying undirected*

[3] The proof of this theorem is deferred to the appendix.

graph. If the treewidth of D' is at most t, then Digraph Bandwidth *can be solved in time* $\mathcal{O}^\star(2^{n+(t+2)\log n})$.

Observe that Theorem 1.5 provides $\mathcal{O}^\star(2^{n+\mathcal{O}(\sqrt{n}\log n)})$ algorithm for directed planar graphs and for digraph whose underlying undirected graph excludes some fixed graph H as a minor. This algorithm in fact, yields a $2^{\mathcal{O}(n)}$ time algorithm even when the treewidth of the underlying undirected graph of the given digraph is $\mathcal{O}(n/\log n)$. Notice that Theorem 1.2 gives $2^{\mathcal{O}(n)}$ time algorithm for digraphs of constant average degree, while Theorem 1.5 will not apply to these cases as these digraphs could contain expander graphs of constant degree whose treewidth of the underlying undirected graph could be n/c, for some fixed constant c. On the other hand Theorem 1.5 could give $2^{\mathcal{O}(n)}$ time algorithm for digraphs that have $\mathcal{O}(n^2/\log n)$ arcs but treewidth is $\mathcal{O}(n/\log n)$. Thus, Theorems 1.2 and 1.5 give $2^{\mathcal{O}(n)}$ time algorithm for different families of digraphs.

2 Preliminaries

The standard notation about sets and functions has been deferred to the Appendix. For positive integers i, j, $[i] = \{1, \cdots, i\}$ and $[i, j] = \{i, \cdots, j\}$. For any set X, by $X = (X_1, X_2)$ we denote an ordered partition of X, that is $X_1 \cup X_2 = X$, $X_1 \cap X_2 = \emptyset$ and, (X_1, X_2) and (X_2, X_1) are two different partitions of X. For any functions $f_1 : X_1 \to Y_1$ and $f_2 : X_2 \to Y_2$, we say that f_1 *is consistent with* f_2 if for each $x \in X_1 \cap X_2$, $f_1(x) = f_2(x)$. If f_1 and f_2 are consistent, then $f_1 \cup f_2 \colon X_1 \cup X_2 \to Y_1 \cup Y_2$ is defined as $(f_1 \cup f_2)(x) = f_i(x)$, if $x \in X_i$. For any set V of size n, we call a function $\sigma \colon V \to [n]$ as an *ordering* of V). Given an ordering σ of $V(D)$, an arc $(u, v) \in E(D)$ is called a *forward arc in* σ if $\sigma(u) < \sigma(v)$, otherwise it is called a *backward arc*. For a natural number $b \in \mathbb{N}$, we call σ as a *b-ordering of D* if for any forward arc $(u, v) \in E(D)$, $\sigma(v) - \sigma(u) \leq b$, that is, if it has digraph bandwidth at most b. Given a set V and an integer b, a b-bucketing of V is a function $\mathtt{B} : V \to [p, q]$, such that $p, q \in \mathbb{N}$ and for each $i \in [p, q-1]$, $|\mathtt{B}^{-1}(i)| = b$ and $|\mathtt{B}^{-1}(q)| \leq b$. Note that, if $|V|$ is a multiple of b, then $\mathtt{B}^{-1}(q) = b$ and $(q - p + 1) \cdot b = |V|$. If for each $i \in [p, q]$, $|\mathtt{B}^{-1}(i)| \leq b$, we call $\mathtt{B}$ a *partial b-bucketing* of V. Note that, for any b, every b-bucketing is a partial b-bucketing. For a (partial) b-bucketing $\mathtt{B} : V \to [p, q]$, we say that an element $v \in V$ is assigned the *i-th bucket of* $\mathtt{B}$ if $\mathtt{B}(v) = i$ and $\mathtt{B}(v)$ is called the *bucket of v*. Also, b is called the *size of the bucket* $\mathtt{B}(v)$. If $\mathtt{B}(u) = i$ and $\mathtt{B}(v) = j$ and $j > i$, then *the number of buckets between the buckets of u and v* is equal to $j - i - 1$. Also, the *number of elements of V in the buckets between i and j is* $(j - i - 1) \cdot b$. In explanations, we sometimes drop b and call $\mathtt{B}$ a *(partial) bucketing* to mean that it is a b-bucketing for some b that should be clear from the context. Given a set V, an integer b and an ordering σ of V, one can associate a b-bucketing with σ which assigns the first b elements in σ the 1-st bucket, the next b-elements the next and so on. This is formalized below. Given a set V, an integer b and an ordering σ of V, we say a b-bucketing $\mathtt{B}$ *respects* σ if $\mathtt{B} : V \to [\lceil |V|/b \rceil]$ is defined as follows. For any $x \in [|V|]$, if $x = ib + j$ for some $i \in \mathbb{N}$ such that $j < b$, then $\mathtt{B}(\sigma_x) = i + 1$ if $j > 0$, and $\mathtt{B}(\sigma_x) = i$ if $j = 0$.

In the upcoming sections, the proofs marked with $\star$ can be found in the Appendix.

3 Exact Algorithm for Directed Bandwidth for dags

The goal of this section is to prove Theorem 1.1. The algorithm follows the ideas of Cygan and Pilipczuk [10]. We give the details here for the sake of completeness and to mention the little details where we deviate from the algorithm of [10]. Throughout this section, without loss of generality, we can assume that the input digraph D is weakly connected, as otherwise,

265 one can solve the problem on each of the weakly connected components of D and concatenate
266 the orderings obtained from each of them, in any order, to get the final ordering. Also,
267 instead of working on the optimization version of the problem, we work on the decision
268 problem, where together with the input digraph D, one is given an integer b, and the goal is
269 to decide whether there exists a topological ordering of $V(D)$ of bandwidth at most b. It is
270 easy to see that designing an algorithm for this decision version with the desired running
271 time is enough to prove Theorem 1.1. In the following, we abuse notation a little and call
272 (D, b) as an instance of DAG-BW.

273 Throughout the remaining section, we call a topological ordering of D of bandwidth b as
274 a b-*topological ordering*. A b-bucketing of $V(D)$ is called a b-*topological bucketing* if for all
275 $(u, v) \in E(D)$, either $\texttt{B}(u) = \texttt{B}(v)$ or $\texttt{B}(v) = \texttt{B}(u)+1$. Our algorithm, like the algorithm of [10],
276 has two phases : Bucketing and Ordering. The Bucketing phase of the algorithm is
277 described by Lemma 3.1.

278 ▶ **Lemma 3.1.** ($\star$) Given an instance (D, b) of DAG-BW, one can find a collection $\mathcal{B}$, of
279 $(b+1)$-topological bucketings of $V(D)$ of size at most $2^{n-1} \cdot \lceil n/b+1 \rceil$, in time $\mathcal{O}^\star(2^n)$, such
280 that for every b-topological ordering σ of D, there exists a bucketing $\texttt{B} \in \mathcal{B}$ such that $\texttt{B}$
281 respects σ.

282 In the Ordering phase, given a $(b+1)$-topological bucketing $\texttt{B}$, the algorithm finds a
283 b-topological ordering σ of D, if it exists, such that $\texttt{B}$ respects σ. From Lemma 3.1, the family
284 $\mathcal{B}$ guarantees the existence of a $(b+1)$-topological bucketing $\texttt{B}$ of the final desired ordering,
285 if it exists. To execute this step, we use the idea of finding a sequence of *lexicographically*
286 *embeddibe sets* using dynamic programming as used in [10]. To define lexicographically
287 embeddible set, the authors first defined the notion of *lexicographic ordering of slots*. We use
288 the same definition in this paper.

289 ▶ **Definition 3.2** (Lexicographic ordering of *slots*). *Given an integer b, let* $\texttt{bucket}\colon [n] \to$
290 $\lceil n/(b+1) \rceil$ *be a function such that* $\texttt{bucket}(i) = \lceil i/(b+1) \rceil$ *and* $\texttt{pos}\colon [n] \to [b+1]$ *be a function*
291 *such that* $\texttt{pos}(i) = ((i-1) \mod (b+1)) + 1$. *We define the lexicographic ordering of slots*
292 *as the lexicographic ordering of* $(\texttt{pos}(i), \texttt{bucket}(i))$*, where* $i \in [n]$.

293 For the Bandwidth problem, the authors of [10] proceed as follows. Given a graph
294 $G = (V, E)$, and a $(b+1)$-bucketing, $\texttt{B}$ of $V(G)$, they prove that there exists an ordering
295 σ of G such that $\texttt{B}$ respects σ if and only if there exists a sequence of subsets of $V(G)$,
296 $\emptyset \subset S_1 \subset \cdots \subset S_n$, $|S_i| = i$, for all $i \in [n]$, such that each S_i satisfies the following properties:
297 (i) for each S_i, there is a mapping $\gamma_{S_i}: S_i \to [n]$ such that $\texttt{bucket}(\gamma_{S_i}(v)) = \texttt{B}(v)$, and the set
298 $\{(\texttt{pos}(\gamma_{S_i}(v)), \texttt{bucket}(\gamma_{S_i}(v))) \mid v \in S_i\}$ is the set of first $|S_i|$ elements in the lexicographic
299 ordering of slots, and (ii) if $u \in S_i$ and $v \notin S_i$, then $\texttt{B}(v) \leq \texttt{B}(u)$. They call such a set S_i
300 as a lexicographically embeddible set. They then obtain $\gamma_{S_{i+1}}$ by extending γ_{S_i} as follows.
301 If $v \in S_i \cap S_{i+1}$, then $\gamma_{S_{i+1}}(v) = \gamma_i(v)$, otherwise $(\texttt{pos}(\gamma_{S_{i+1}}(v)), \texttt{bucket}(\gamma_{S_{i+1}}(v)))$ is the
302 $|S_{i+1}|^{\text{th}}$ element in the lexicographic embedding of slots. Recall that there is only one vertex
303 v in $S_{i+1} \setminus S_i$. Furthermore, if v has a neighbor u in S_i, then $\texttt{B}(v) \leq \texttt{B}(u)$. If $\texttt{B}(v) = \texttt{B}(u)$,
304 then since bucket size is at most $b+1$, $|\gamma(v) - \gamma(u)| \leq b$. If $\texttt{B}(v) < \texttt{B}(u)$, then by construction
305 of $\gamma_{S_{i+1}}$, $\texttt{pos}(\gamma_{S_{i+1}}(v)) > \texttt{pos}(\gamma_{S_{i+1}}(u))$. Now, again since each bucket size is at most $b+1$,
306 $|\gamma(v) - \gamma(u)| \leq b$. Therefore, γ_{S_i} can be extended to $\gamma_{S_{i+1}}$. Thus, γ_{S_n} will yield the final
307 ordering. Hence, the goal reduces to finding a sequence $S_1 \subset \cdots \subset S_n$, $|S_i| = i$, for all $i \in [n]$,
308 such that each S_i is a lexicographically embeddible set. We will call such a sequence as a
309 *lexicographically embeddible sequence* (les, in short).

310 We proceed in a similar way for Directed Bandwidth. We first note that one cannot
311 use the same definition of lexicographically embeddible set as defined above due to the

following reason. Suppose that S_i is a lexicographical embeddible set. Consider a vertex $u \in S_i$. Suppose that there exists a vertex $v \notin S_i$ that is an in-neighbor of u, then v cannot belong to the bucket of u as it will not lead to the topological ordering using the method defined above. Also, if v is an out-neighbor of u, then since we are working with topological bucketing, v does not belong to the bucket that precedes the bucket of u. Hence, v belongs to the bucket of u. We also want γ as a topological ordering. Therefore, we redefine the notion of lexicographically embeddible set for Directed Bandwidth as follows.

▶ **Definition 3.3** (Lexicographically embeddible set for digraphs)**.** *Given a $(b+1)$-topological bucketing $\mathtt{B}$, of $V(D)$, we say that $S \subseteq V(D)$ is a lexicographically embedibble set if the following condition holds.*

(C1) *For each arc $(u,v) \in E(D)$ such that $u \in S$ and $v \notin S$, $\mathtt{B}(u) = \mathtt{B}(v)$.*

(C2) *For each arc $(u,v) \in E(D)$ such that $v \in S$ and $u \notin S$, $\mathtt{B}(u) = \mathtt{B}(v) - 1$.*

(C3) *There exists a b-topological ordering $\gamma\colon S \to [n]$ such that for all $v \in S$, $\mathtt{bucket}(\gamma(v)) = \mathtt{B}(v)$, and $(\mathtt{pos}(\gamma(v)), \mathtt{bucket}(\gamma(v)))$ belongs to the first $|S|$ elements in the lexicographic ordering of slots. We refer γ as a partial b-topological ordering that respects lexicographic ordering of slots.*

given a $(b+1)$-topological bucketing $\mathtt{B}$ of $V(D)$, to find a b-topological ordering σ such that $\mathtt{B}$ respects σ,

▶ **Lemma 3.4.** ($\star$) Given a $(b+1)$-topological bucketing $\mathtt{B}$ of $V(D)$, the following are equivalent. **(i)** There exists a b-topological ordering σ of the digraph D such that the unique $(b+1)$-bucketing induced by σ, that is $\mathtt{B}_\sigma$, is $\mathtt{B}$. In other words, $\mathtt{B}_\sigma(v) = \mathtt{B}(v)$, for all $v \in V(D)$, **(ii)**There exists a les, $\emptyset \subset S_1 \subset \cdots \subset S_n = V$.

Due to Lemma 3.4, our goal is reduced to find a les. Cygan and Pilipczuk [10] find les using dynamic programming over subsets of the vertex set of given graph. We use a similar dynamic programming approach with appropriate modification because of the revised definition of a lexicographically embeddible set. In the dynamic programming table, for each $S \subseteq V(D)$, $c[S] = 1$, if and only if S is a lexicographically embeddible set. To compute the value of $c[S]$, we first find a vertex $v \in S$ such that $S \setminus \{v\}$ is a lexicographically embeddible set, that is, $c[S \setminus \{v\}] = 1$, and v satisfies the following properties : (i) for all the arcs $(v,u) \in E(D)$ such that $u \notin S$, $\mathtt{B}(v) = \mathtt{B}(u)$; (ii) for all the arcs $(u,v) \in E(D)$ such that $u \notin S$, $\mathtt{B}(u) = \mathtt{B}(v) - 1$; and (iii) $\mathtt{B}(v) = ((|S|-1) \mod \lceil n/(b+1) \rceil) + 1$. We compute the value of $c[S]$ for every subset $S \subseteq V(D)$. Note that if $c[V(D)] = 1$, then $V(D)$ is a lexicographically embeddible set. Also, we can compute les by backtracking in the dynamic programming table.

▶ **Lemma 3.5.** ($\star$) Given an instance (D,b) of DAG-BW, and a $(b+1)$-topological bucketing $\mathtt{B}$ of $V(D)$ that respects some b-topological ordering of D (if it exists), one can compute a les in time $\mathcal{O}^\star(2^n)$.

Note that using Lemmas 3.1, 3.4 and 3.5, one can solve DAG-BW in $\mathcal{O}^\star(4^n)$ time. The desired running time of $\mathcal{O}^\star(3^n)$ in Theorem 1.1 can be proved by careful analysis of two steps in the algorithm as done in Theorem 12 of [10]. Since the proof of this is the same as Theorem 12 of [10], we defer the proof here.

4 Exact Algorithm for Digraph Bandwidth via Directed Bandwidth

We call an ordering of the vertex set of a digraph a *b-ordering* if its digraph bandwidth is at most b. In order to prove Theorem 1.2 observe the following. Let (D, b) be an instance of (the decision version of) Digraph Bandwidth. If it is a Yes instance, then let σ be a b-ordering of D. Let R be the set of backward arcs in σ. Note that σ is a topological ordering of $D - R$. If we guess the set of backward arcs R in a b-ordering of D (which takes time 2^m), then the goal is reduced to finding a b-topological ordering, σ, of $D - R$ such that if $(u, v) \in R$, then $\sigma(u) > \sigma(v)$. In fact, one can also observe that it is sufficient to find a b-topological ordering, ρ, of $D - R$ such that for all $(u, v) \in R$ either $\rho(u) > \rho(v)$ or $\rho(v) - \rho(u) \leq b$. We claim that we can find the required ordering of $D - R$ using the algorithm for Directed Bandwidth for dags given in Section 3. Suppose that σ is a b-ordering of D. Let $\mathtt{B}_\sigma$ be a $(b+1)$-bucketing that respects σ. Let R be the set of backward arcs in σ. Since σ is a b-topological ordering of $D - R$, using Lemma 3.1, $\mathtt{B}_\sigma$ belongs to the collection of $(b + 1)$-bucketings $\mathcal{B}$. Now, using Lemmas 3.5 and 3.4, we obtain a b-topological ordering ρ of $D - R$ that respects $\mathtt{B}_\sigma$. Note that ρ is a b-ordering of D, as for each arc $(u, v) \notin R$, $\rho(v) - \rho(u) \leq b$ because ρ is a b-topological ordering of $D - R$. Also, if $(u, v) \in R$, then observe that if $\mathtt{B}_\sigma(u) \neq \mathtt{B}_\sigma(v)$, then since both σ and ρ respect $\mathtt{B}_\sigma$ and (u, v) is a backward arc in σ, thus, (u, v) is a backward arc in ρ too, that is, $\rho(u) > \rho(v)$. Otherwise, if $(u, v) \notin R$ and $\mathtt{B}_\sigma(u) = \mathtt{B}_\sigma(v)$, then since ρ respects $\mathtt{B}_\sigma$ and the size of the buckets of $\mathtt{B}_\sigma$ is $(b + 1)$, therefore, $|\rho(u) > \rho(v)| \leq b$. Thus, the algorithm of Theorem 1.2 runs the algorithm for DAG-BW for each $R \subseteq V(D)$, to obtain the desired running time.

5 (Single) Exponential Time Approximation Scheme for Digraph Bandwidth

The goal of this section is to prove Theorems 1.3 and 1.4. Let (D, b) be an instance of (the decision version of) Digraph Bandwidth. The algorithm relies on an interesting property of a b-bucketing that respects a b-ordering of D. Let σ be a b-ordering of D and let $\mathtt{B}$ be a b-bucketing of $V(D)$ that respects σ. An interesting property of such a bucketing $\mathtt{B}$ is that *if* $(u, v) \in E(D)$, *then either* $\mathtt{B}(u) > \mathtt{B}(v)$ *or* $\mathtt{B}(v) \leq \mathtt{B}(u) + 1$. This is because the size of each bucket in $\mathtt{B}$ is b and σ is a b-ordering of D. Let us call this property of a b-bucketing *useful*. What we saw in the previous section is that if we somehow have a bucketing that respects σ, then one can design an algorithm to fetch σ from this bucketing. In this section, instead of seeking for a bucketing that respects σ we seek for a bucketing with the above mentioned useful property. Observe that, while the existence of such a bucketing with this useful property might not necessarily imply the existence of some b-ordering of D, but having such a bucketing with, for example buckets of size b, definitely yields a $2b$-ordering of D. This is because, given such a bucketing one can assign positions to vertices in the ordering by choosing any arbitrary ordering amongst the vertices that belong to the same bucket and concatenating these orderings in the order of the bucket numbers. By changing the bucket size in the described bucketing, one can yield an ordering of digraph bandwidth at most $(1 + \epsilon)$ times the optimal. This procedure, as we will see, also give an optimal digraph bandwidth ordering when we use the bucket sizes to be 1. Below we give the description of the algorithm used to find a bucketing with the useful property.

We begin by formulating the useful property of a bucketing described above. Since the size of buckets is uniform in a bucketing, instead of defining the property in terms of

bucket numbers, we describe it in terms of the number of vertices that can appear between the two buckets corresponding to the end points of a forward arc in the ordering. Such a shift in definition helps us to get slightly better bounds in our running time. For any positive integers b, s and a digraph D, given $X \subseteq V(D)$ and a (partial) b-bucketing of X, say $\mathtt{B} : X \to [p, q]$ for some $p, q \in \mathbb{N}$, we say that the *external stretch of* $\mathtt{B}$ *is at most* s if for each arc $(u, v) \in E(D[X])$, either $\mathtt{B}(u) \geq \mathtt{B}(v)$, or $(\mathtt{B}(v) - \mathtt{B}(u) - 1) \cdot b \leq s$. Recall that $\mathtt{B}(u) - \mathtt{B}(v) - 1$ denote the number of buckets between the bucket of u and the bucket of v. Our major goal now is to prove Lemma 5.1.

▶ **Lemma 5.1.** *Given a digraph D and positive integers b, s, there is an algorithm, that runs in time* $\min\{\mathcal{O}^*(4^n \cdot (\lceil {s+1}/{b} \rceil)^n), \mathcal{O}^*(4^n \cdot (\lceil {s+1}/{b} \rceil)^{2(b+s)\log n})\}$*, and computes a b-bucketing of $V(D)$, $\mathtt{B} : V(D) \to [\lceil {|V(D)|}/{b} \rceil]$, of external stretch at most s.*

We give a recursive algorithm for Lemma 5.1 (Algorithm 1). Since Algorithm 1 is recursive, the input of the algorithm will contain a few more things in addition to D, b, s to maintain the invariants at the recursive steps. We give the description of the input to Algorithm 1 in Definition 5.2.

▶ **Definition 5.2** (Legitimate input for Algorithm 1)**.** *The input $(D, b, s, first, last, left\text{-}bor(V(D)), right\text{-}bor(V(D)), \mathtt{B}_{in})$ is called* legitimate *for Algorithm 1 if the following holds. Let $\delta = \lceil {s+1}/{b} \rceil$.*

(P1) D is a digraph, b, s are positive integers and $|V(D)| = 2^\eta \cdot b \cdot \delta$, where $\eta \geq 0$ is a positive integer.
(P2) $first$ and $last$ are positive integers such that $last - first + 1 = 2^\eta$, where η is such that $|V| = 2^\eta \cdot b \cdot \delta$.
(P3) $left\text{-}bor(V(D)), right\text{-}bor(V(D)) \subseteq V(D)$,
(P4) $\mathtt{B}_{in} : left\text{-}bor(V(D)) \cup right\text{-}bor(V(D)) \to [first, last]$ is a partial b-bucketing such that for each $v \in left\text{-}bor(V(D))$, $\mathtt{B}_{in}(v) \in [first, first + \delta - 1]$, for each $v \in right\text{-}bor(V(D))$, $\mathtt{B}_{in}(v) \in [last - \delta + 1, last]$ and the external stretch of $\mathtt{B}_{in}$ is at most s.

Observe that $\delta - 1$ represents the number of buckets that can appear between the buckets of u and v in any b-bucketing of external stretch at most s, where the bucket of u precedes the bucket of v and $(u, v) \in E(D)$. We would like to remark that the condition of (P1) is not serious as we could have worked without it. We state it like the way we do for the sake of notational and argumentative convenience in the proofs. All it states is that the number of vertices is a power of 2 multiplied by b and δ. The $first$ and $last$ in (P2) represents the bucket number of the first and last buckets in the bucketing to be outputted. The relation between $first$ and $last$ in (P2) is there to ensure that there are enough buckets to hold the vertices of D. At any recursive call, the sets $left\text{-}bor(V(D))$ and $right\text{-}bor(V(D))$ represent the sets of vertices whose buckets have already been fixed in the previous recursive calls. The set $left\text{-}bor(V)$ represents the set of vertices in V that have an in-neighbour to the vertices that have been decided to be placed in the buckets before the bucket numbered $first$ in the earlier recursive calls. Similarly, $right\text{-}bor(V)$ represents the set of vertices in V that have an out-neighbour to the vertices that have been decided to be placed in the buckets after the bucket numbered $last$ in the earlier recursive calls. Thus, in order to give the final bucketing of external stretch at most s, it is necessary that $left\text{-}bor(V)$ are placed in the first δ buckets and $right\text{-}bor(V)$ are placed in the last δ buckets. This is captured in (P4). The next definition describes the properties of the bucketing that would be outputted by Algorithm 1.

443 ▶ **Definition 5.3** (Look-out bucketing for a legitimate instance). *Given a legitimate instance* $\mathcal{I} = (D, b, s, first, last, left\text{-}bor(V(D)), right\text{-}bor(V(D)), \mathtt{B}_{in})$*, we say a bucketing* $\mathtt{B}$ *is a* look-out bucketing for $\mathcal{I}$*, if* $\mathtt{B}$ *is a* b*-bucketing* $\mathtt{B}_{out} : V \to [first, last]$ *of external stretch at most* b *that is consistent with* $\mathtt{B}_{in}$*.*

447 Observe that, for the algorithm of Lemma 5.1, a call to Algorithm 1 on $(D, b, s, 1, |V|/b, \emptyset, \emptyset, \phi)$ is enough. The formal proof of correctness for the same can be found in the Appendix. To give the formal description of Algorithm 1, we will use the following definition.

450 ▶ **Definition 5.4** ($\mathtt{B}$ validates a partition (X_1, X_2)). *For any integers* p, q *and* $X' \subseteq X$*, let* $\mathtt{B} : X' \to [p, q]$ *be a partial bucketing of* X'*. Let* (X_1, X_2) *be some partition of* X*. We say that* $\mathtt{B}$ validates (X_1, X_2) *if the following holds. Let* $r = \lfloor (p+q)/2 \rfloor$*. For each* $v \in X_1 \cap X'$*,* $\mathtt{B}(v) \in [p, r]$ *and for each* $v \in X_2 \cap X'$*,* $\mathtt{B}(v) \in [r + 1, q]$*.*

Algorithm 1 Algorithm for computing b-bucketing of external stretch at most s

Input: $\mathcal{I} = (D, b, s, first, last, left\text{-}bor(V), right\text{-}bor(V), B_{in})$ such that $\mathcal{I}$ is legitimate for Algorithm 1.
Output: A look-out bucketing for $\mathcal{I}$, if it exists.

1: Let $V = V(D)$ and $\delta = \lceil s+1/b \rceil$.
2: **if** $|V| = b \cdot \delta$ **then**
3: **return** any b-bucketing $\mathtt{B} : V \to [first, last]$ that it consistent with $\mathtt{B}_{in}$
4: Let $mid = (first+last-1)/2$.
5: **for** each partition (L, R) of V such that $|L| = |R|$ and B_{in} validates (L, R) **do**
6: Let $bor_L = \{v \in L \mid$ there exists $u \in R, (v, u) \in E(D)\}$.
7: Let $bor_R = \{v \in R \mid$ there exists $u \in L, (u, v) \in E(D)\}$.
8: Let $\mathcal{B}$ be the collection of partial b-bucketings, $\mathtt{B} : bor_L \cup bor_R \to [mid - \delta + 1, mid + \delta]$, such that for each $v \in bor_L$, $\mathtt{B}(v) \in [mid - \delta + 1, mid]$, for each $v \in bor_R$, $\mathtt{B}(v) \in [mid + 1, mid + \delta]$, external stretch of $\mathtt{B}$ is at most s and $\mathtt{B}$ is consistent with $\mathtt{B}_{in}$.
9: **for** each $\mathtt{B} \in \mathcal{B}$ **do**
10: Define $\mathtt{B}^{new}_{in} : left\text{-}bor(V) \cup bor_L \cup bor_R \cup right\text{-}bor(V) \to [first, last]$, such that for each $v \in left\text{-}bor(V) \cup right\text{-}bor(V)$, $\mathtt{B}^{new}_{in}(v) = \mathtt{B}_{in}(v)$ and, for each $v \in bor_L \cup bor_R$, $\mathtt{B}^{new}_{in}(v) = \mathtt{B}(v)$.
11: Let $\mathtt{B}^{newL}_{in} : left\text{-}bor(V) \cup bor_L \to [first, mid]$ be such that $\mathtt{B}^{newL}_{in} = \mathtt{B}^{new}_{in}{}_{|L}$.
12: Let $\mathtt{B}^{newR}_{in} : bor_R \cup right\text{-}bor(V) \to [mid + 1, last]$ be such that $\mathtt{B}^{newR}_{in} = \mathtt{B}^{new}_{in}{}_{|R}$.
13: Define $left\text{-}bor(L) = left\text{-}bor(V)$ and $right\text{-}bor(L) = bor_L$.
14: Define $left\text{-}bor(R) = bor_R$ and $right\text{-}bor(R) = right\text{-}bor(V)$.
15: Let $\mathcal{I}^{\mathtt{B}}_L$ be the instance $(D[L], b, s, first, mid, left\text{-}bor(L), right\text{-}bor(L), \mathtt{B}^{newL}_{in})$.
16: Let $\mathcal{I}^{\mathtt{B}}_R$ be the instance $(D[R], b, s, mid, last, left\text{-}bor(R), right\text{-}bor(R), \mathtt{B}^{newR}_{in})$.
17: **if** $\mathcal{I}^{\mathtt{B}}_L$ and $\mathcal{I}^{\mathtt{B}}_R$ are legitimate inputs for Algorithm 1 **then**
18: **if** *Algorithm* $1(\mathcal{I}^{\mathtt{B}}_L)$! = NO and *Algorithm* $1(\mathcal{I}^{\mathtt{B}}_R)$! = NO **then**
19: **return** Algorithm 1 $(\mathcal{I}^{\mathtt{B}}_L) \cup$ Algorithm 1 $(\mathcal{I}^{\mathtt{B}}_R)$
20: **return** NO

454 For the formal description of Algorithm 1 refer to the pseudocode. We give the informal description of Algorithm 1 here. (Figure 1 for the same can be found in Appendix). In a legitimate instance when the number of vertices is $b \cdot \delta$, the number of buckets is δ. Recall that $\delta = \lceil s+1/b \rceil$. Note that in this case, every b-bucketing of the vertex set has external stretch at most s. This is because the number of buckets between any two buckets is at most

$\delta - 2$ and hence, the number of vertices that appear in the buckets between any two buckets is at most $(\delta - 2) \cdot b \leq s$.

When the number of vertices is larger, the algorithm first guesses which vertices will be assigned a bucket from the first half buckets of the final bucketing (this corresponds to the set L) and which will be assigned the last half (this corresponds to the set R). Since the final bucketing has to be consistent with $\mathtt{B}_{in}$, from the description of $\mathtt{B}_{in}$ in Definition 5.2, the vertices of $left\text{-}bor(V)$ should belong to the first half buckets and the vertices of $right\text{-}bor(V)$ should belong to the last half buckets. Thus, the algorithm only considers those partitions (guesses) which $\mathtt{B}_{in}$ validates (Line 5).

Fix a guessed partition (L, R) of V. The set bor_L represents the set of vertices in L that have an out-neighbour in R. Similarly, the set bor_R represents the set of vertices in R with an in-neighbour in L. Since in any b-bucketing of external stretch at most s, the number of buckets that can appear between the buckets of the end points of a forward arc is at most $\delta - 1$, the vertices of bor_L can only be placed in the δ buckets closest to the middle bucket and before it. Similarly, the vertices of bor_R can only be placed in the δ buckets closest to the middle bucket and after it. The algorithm goes over all possible partial b-bucketings of these vertices in the described buckets, that are consistent with $\mathtt{B}_{in}$, and themselves have external stretch at most δ (Line 8).

For a fixed partial b-bucketing enumerated above, the algorithm recursively finds a bucketing of the L vertices in the first half buckets and the bucketing of the R vertices in the last half buckets that is consistent with $\mathtt{B}_{in}$ and the partial b-bucketing of the bor_L and bor_R vertices guessed. This final bucketing is then obtained by combing the two bucketings from the two disjoint sub-problems (Lines 9 to 19).

▶ **Lemma 5.5.** ($\star$) Algorithm 1 on a legitimate input $(D, b, s, first, last, left\text{-}bor(V), right\text{-}bor(V), B_{in})$, runs in time $\min\{\mathcal{O}^*(4^n \cdot \lceil s+1/b \rceil^n), \mathcal{O}^*(4^n \cdot \lceil s+1/b \rceil^{2(b+s)\log n})\}$, and returns a look-out bucketing for $\mathcal{I}$, if it exists.

Theorem 1.3 (resp. Theorem 1.3) can be proved by setting bucket size to be $\lceil b\epsilon/2 \rceil$ (resp. 1) and external stretch $b - 1$ as parameters in the algorithm of Lemma 5.1. The full proofs are deferred to the Appendix.

6 Conclusion

In this paper we gave exponential time algorithm for the Digraph Bandwidth problem, that either solved the problem exactly or computed it approximately. In particular, our results imply that whenever $b \leq \frac{n}{\log^2 n}$ or, the treewidth of the underlying undirected digraph is $\mathcal{O}(\frac{n}{\log n})$ or, the number of arcs in the digraph are linear in the number of vertices, then there exists a $2^{\mathcal{O}(n)}$ time algorithm for solving Digraph Bandwidth. Some important questions that remain open about Digraph Bandwidth are the following.

- Does Digraph Bandwidth admit an algorithm with running time $2^{\mathcal{O}(n)}$ on general digraphs?
- Another interesting question is the complexity of the Digraph Bandwidth problem, when b is fixed. Recall that, in the undirected case, Bandwidth can be solved in time $\mathcal{O}(n^{b+1})$ [26]. When $b = 0$, the problem is equivalent to checking if the input is a dag, which can be done in linear time. For $b = 1$, we are able to design an $\mathcal{O}(n^2)$ time algorithm. For $b = 2$, the problem seems to be extremely complex, and in fact, we will be surprised if the problem turns out to be polynomial time solvable. Overall, finding the complexity of Digraph Bandwidth, for a fixed $b \geq 2$, is an interesting open problem.

504 ## References

505 **1** O. Amini, F.V. Fomin, and S. Saurabh. Counting subgraphs via homomorphisms. *SIAM J.*
506 *Discrete Math.*, 26(2):695–717, 2012.
507 **2** S.F. Assmann, G.W. Peck, M.M. Sysło, and J. Zak. The bandwidth of caterpillars with hairs
508 of length 1 and 2. *SIAM J. Alg. Discrete Meth.*, 2(4):387–393, 1981.
509 **3** G. Blache, M. Karpiński, and J. Wirtgen. *On approximation intractability of the bandwidth*
510 *problem.* Citeseer, 1997.
511 **4** H.L. Bodlaender, M.R. Fellows, and M.T. Hallett. Beyond NP-completeness for problems of
512 bounded width (extended abstract): hardness for the W hierarchy. In *Proc. of STOC 1994*,
513 pages 449–458, 1994.
514 **5** M. Chudnovsky, A. Fradkin, and P. Seymour. Tournament immersion and cutwidth. *J. Comb.*
515 *Theory Ser. B*, 102(1):93–101, 2012.
516 **6** M. Cygan and M. Pilipczuk. Faster exact bandwidth. In *Proc. of WG 2008.*
517 **7** M. Cygan and M. Pilipczuk. Exact and approximate bandwidth. *Theor. Comput. Sci.*,
518 411(40-42):3701–3713, 2010.
519 **8** M. Cygan and M. Pilipczuk. Bandwidth and distortion revisited. *Discrete Appl. Math.*,
520 160(4-5):494–504, 2012.
521 **9** M. Cygan and M. Pilipczuk. Even faster exact bandwidth. *ACM Trans. Algorithms*, 8(1):8:1–
522 8:14, 2012.
523 **10** Marek Cygan and Marcin Pilipczuk. Faster exact bandwidth. In *International Workshop on*
524 *Graph-Theoretic Concepts in Computer Science*, pages 101–109. Springer, 2008.
525 **11** J. Díaz, M. Serna, and D.M. Thilikos. Counting h-colorings of partial k-trees. *Theor. Comput.*
526 *Sci.*, 281(1-2):291–309, 2002.
527 **12** M.S. Dregi and D. Lokshtanov. Parameterized complexity of bandwidth on trees. In *Proc. of*
528 *ICALP 2014*, pages 405–416, 2014.
529 **13** U. Feige. Coping with the NP-hardness of the graph bandwidth problem. In *Proc. of SWAT*
530 *2000*, pages 10–19. Berlin, 2000.
531 **14** U. Feige and K. Talwar. Approximating the bandwidth of caterpillars. In *Proc. of APPROX-*
532 *RANDOM 2005*, volume 3624, pages 62–73, 2005.
533 **15** M. Fürer, S. Gaspers, and S.P. Kasiviswanathan. An exponential time 2-approximation
534 algorithm for bandwidth. *Theor. Comput. Sci.*, 511:23–31, 2013.
535 **16** M. Garey, R. Graham, D. Johnson, and D. Knuth. Complexity results for bandwidth
536 minimization. *SIAM J. Appl. Math.*, 34(3):477–495, 1978.
537 **17** M.R. Garey and D.S. Johnson. *Computers and intractability*, volume 29. wh freeman New
538 York, 2002.
539 **18** P.A. Golovach, P. Heggernes, D. Kratsch, D. Lokshtanov, D. Meister, and S. Saurabh. Band-
540 width on AT-free graphs. *Theor. Comput. Sci.*, 412(50):7001–7008, 2011.
541 **19** P. Heggernes, D. Kratsch, and D. Meister. Bandwidth of bipartite permutation graphs in
542 polynomial time. *J. Discrete Algorithms*, 7(4):533–544, 2009.
543 **20** D.J. Kleitman and R. Vohra. Computing the bandwidth of interval graphs. *SIAM J. Discrete*
544 *Math.*, 3(3):373–375, 1990.
545 **21** R. Krauthgamer, J.R. Lee, M. Mendel, and A. Naor. Measured descent: A new embedding
546 method for finite metrics. In *Proc. of FOCS 2004*, pages 434–443, 2004.
547 **22** Yung-Ling Lai and Kenneth Williams. A survey of solved problems and applications on
548 bandwidth, edgesum, and profile of graphs. *Journal of graph theory*, 31(2):75–94, 1999.
549 **23** Marek M. Karpinski, Jürgen Wirtgen, and A. Zelikovsky. An approximation algorithm for the
550 bandwidth problem on dense graphs. Technical report, 1997.
551 **24** B. Monien. The bandwidth minimization problem for caterpillars with hair length 3 is
552 NP-complete. *SIAM J. Alg. Discrete Meth.*, 7(4):505–512, 1986.
553 **25** C.H. Papadimitriou. The NP-completeness of the bandwidth minimization problem. *Computing*,
554 16(3):263–270, 1976.

555 **26** J.B. Saxe. Dynamic-programming algorithms for recognizing small-bandwidth graphs in
556 polynomial time. *SIAM J. Alg. Discrete Meth.*, 1(4):363–369, 1980.
557 **27** W. Unger. The complexity of the approximation of the bandwidth problem. In *Proc. of FOCS*
558 *1998*, pages 82–91, 1998.
559 **28** V. Vassilevska, R. Williams, and S.L.M. Woo. Confronting hardness using a hybrid approach.
560 In *Proc. of SODA*, pages 1–10, 2006.
561 **29** J.H. Yan. The bandwidth problem in cographs. *Tamsui Oxford J. Math. Sci*, 13:31–36, 1997.

A A potential scenario that could be modelled by Digraph Bandwidth

Consider the following scenario. Suppose a factory is being set up at a hilly terrain and its different departments need to be placed at different places on an uphill. Each department manufactures particular kinds of tools and there are interdependencies with respect to tools among departments, that is, a department may need a tool which is manufactured by some other department. Suppose that electric vehicles are used to transport the tools from one department to another. Most of the electric vehicles (with few exceptions) can cover approximately 100 miles in a charge which is even less while going uphill. Also, recharging vehicle takes time. On the other hand, while going downhill, either the vehicle consumes less power, or does not consume any power, or even recharge vehicle, depending on the steepness of the hill. Therefore, if the department v requires tools from the department u, then the factory owner would like to locate v either downhill from u, or not very far from u to save the energy consumption. This can be modelled as Digraph Bandwidth as follows. In the digraph D, each vertex represents a department and an arc from u to v means that u requires deliveries from v. Let u and v be two vertices in the digraph D. Then, we have an arc from u to v in the digraph D if the department v uses tools manufactured by department u. Interpret an ordering of vertices from left to right as locations of department from downhill to uphill, that is, the first vertex in the ordering corresponds to the bottommost department at the hill and the last vertex in the ordering corresponds to the topmost department at the hill. Clearly, if directed bandwidth of D is at most b, then if the department u needs tools from the department v, and u is uphill from v, then u is at most b distance away from v (considering unit distance between the departments).

B Some More Preliminaries

Sets and Functions: We denote the set of natural numbers by $\mathbb{N}$. For $i, j \in \mathbb{N}$, $[i]$ and $[i, j]$ denote the sets $\{1, \cdots, i\}$ and $\{i, \cdots, j\}$, respectively. For any set X, by $X = (X_1, X_2)$ we denote an ordered partition of X, that is $X_1 \cup X_2 = X$, $X_1 \cap X_2 = \emptyset$ and, (X_1, X_2) (X_2, X_1) are two different partitions of X. Let $f\colon X \to Y$ be a function. For $y \in Y$, $f^{-1}(y) = \{x \in X \mid f(x) = y\}$. The function f is called injective if for each $x, y \in X$, $f(x) = f(y)$ implies $x = y$. The function $f : \emptyset \to Y$ is denoted by ϕ. For $S \subseteq X$, $f|_S\colon S \to Y$ is a function such that for $s \in S$, we have $f|_S(s) = f(s)$. For any functions $f_1 : X_1 \to Y_1$ and $f_2 : X_2 \to Y_2$, we say that f_1 *is consistent with* f_2 if for each $x \in X_1 \cap X_2$, $f_1(x) = f_2(x)$. If f_1 and f_2 are consistent, then $f_1 \cup f_2\colon X_1 \cup X_2 \to Y_1 \cup Y_2$ is defined as $(f_1 \cup f_2)(x) = f_i(x)$, if $x \in X_i$.

Digraphs: Consider a digraph D. By $V(D)$ and $E(D)$, we denote the set of vertices and (arcs) edges in D, respectively. Throughout the paper, $n = |V(D)|$ and $m = |E(D)|$. A digraph D is called *weakly connected*, if the underlying undirected graph is connected. For any $(u, v) \in E(D)$, u is an *in-neighbour of* v and v is an *out-neighbour* of u. For any set V of size n, we call a function $\sigma\colon V \to [n]$ as an *ordering* of V). Given a function σ, we call $\sigma(u)$ as the *position of* u *in* σ. Given an ordering σ of $V(D)$, an arc $(u, v) \in E(D)$ is called a *forward arc in* σ if $\sigma(u) < \sigma(v)$. For a natural number $b \in \mathbb{N}$, we call σ as a *b-ordering of* D if for any forward arc $(u, v) \in E(D)$, $\sigma(v) - \sigma(u) \leq b$. An ordering σ of $V(D)$ is called a *topological ordering* if every arc $(u, v) \in E(D)$ is a forward arc in σ. A digraph D is called a directed acyclic graph (DAG), if D has a topological ordering. Let σ be an ordering of $V(D)$. By σ_i, we denote the vertex $\sigma^{-1}(i)$. Given two orderings $\sigma_1\colon V(D_1) \to [n_1]$ and $\sigma_2\colon V(D_2) \to [n_2]$,

607 we define the *concatenation of* σ_1 *and* σ_2 as the ordering $\sigma_1 \cdot \sigma_2 \colon V(D_1) \cup V(D_2) \to [n_1 + n_2]$
608 where $\sigma_1(u)$.

609 ▶ **Definition B.1** (Tree decomposition). *A* tree decomposition *of a graph G is a pair $\mathcal{T} =$*
610 *(T, X), where T is a rooted tree and $X = \{X_t \mid t \in V(T)\}$. Every node t of T is assigned a*
611 *subset $X_t \subseteq V(H)$ such that following conditions are satisfied:*

612 - $\bigcup_{t \in V(T)} X_t = V(G)$, *i.e. each vertex in G is in at least one bag;*
613 - *For every edge $uv \in E(G)$, there is $t \in V(T)$ such that $u, v \in X_t$;*
614 - *For every vertex $v \in V(G)$ the graph $T[\{t \in V(T) \mid v \in X_t\}]$ is a connected subtree of T.*

615 The *width* of tree decomposition $\mathcal{T}$ is $\max_{t \in V(T)} |X_t| - 1$, The *treewidth* of a graph G is the
616 minimum possible width of a tree decomposition of G.

617 # C Missing algorithm and proofs of Section 3

618 ## C.1 Description of the algorithm of Lemma 3.1

619 In this section, we give the pseudocode of the algorithm to construct a collection of $(b+1)$-
620 topological bucketings (Algorithm 2). In Algorithm 2, $\mathtt{B}(v) = \star$ denotes that we have not
621 assigned a bucket to v. In Step 3 of Algorithm 2, *queue* is an ordered set. In Step 18 of
622 Algorithm 2, we add a vertex at the end of ordered set *queue*. In Step 20 of Algorithm 2, we
623 remove the first element of *queue*.

624 ## C.2 Proof of Lemma 3.1

625 Suppose that (D, b) is a YES instance of DAG-BW. Let σ be a b-topological ordering of (D, b).
626 Let $\mathtt{B}_\sigma$ be the unique bucketing induced by σ. We compute a collection $\mathcal{B}$, of $(b+1)$-bucketings
627 of D using Algorithm 2. We claim that at each iteration of the algorithm, there exists a
628 bucketing $\mathtt{B} \in \mathcal{B}$ such that if $\mathtt{B}(v) \neq \star$, then $\mathtt{B}(v) = \mathtt{B}_\sigma(v)$. We prove it by induction on the
629 number of iterations. Note that in the first iteration of the algorithm, $\mathcal{B}$ contains all possible
630 $(b+1)$-bucketings of v. Therefore, there exists a bucketing $\mathtt{B} \in \mathcal{B}$ such that $\mathtt{B}(v) = \mathtt{B}_\sigma(v)$.
631 Now, let X be a set of vertices whose bucket has been decided till $(i+1)^{\text{th}}$ iterations of the
632 algorithm. Let x be a vertex that is placed at the $(i+1)^{\text{th}}$ iteration of the algorithm. Let q be
633 the element of *queue* corresponding to which we execute Step 6 of the algorithm for x. Note
634 that if $x \in N^+(q)$, then $x \in \{\mathtt{B}_\sigma(q), \mathtt{B}_\sigma(q) + 1\}$, otherwise $x \in \{\mathtt{B}_\sigma(q), \mathtt{B}_\sigma(q) - 1\}$. Note that
635 we have constructed all these bucketings for x in Step 9 to 17 of the algorithm. By induction
636 hypothesis, there exists $\mathtt{B} \in \mathcal{B}$ such that $\mathtt{B}(u) = \mathtt{B}_\sigma(u)$ for all $u \in X \setminus \{x\}$. Hence, there exists
637 $\mathtt{B} \in \mathcal{B}$ such that $\mathtt{B}(u) = \mathtt{B}_\sigma(u)$ for all $u \in X$. Note that since D is a weakly connected digraph,
638 when *queue* $= \emptyset$, there does not exist $v \in V(D)$ such that $\mathtt{B}(v) = \star$, $\mathtt{B} \in \mathcal{B}$. Now, we prove
639 that $|\mathcal{B}| = \mathcal{O}^\star(2^n)$. Note that in Step 2 of the algorithm, we construct $n/(b+1)$ bucketings for
640 v. Then, for each $u \in V(D) \setminus \{v\}$, we construct at most two $(b+1)$-bucketings. Therefore,
641 $|\mathcal{B}| \leq 2^{n-1} \cdot n/(b+1)$. Since each step of the algorithm can be executed in polynomial time,
642 the running time of the algorithm is $\mathcal{O}^\star(2^n)$ time. ◀

643 ## C.3 Proof of Lemma 3.4

644 Suppose that there exists a b-topological ordering σ of the digraph D such that $\mathtt{B}_\sigma(v) = \mathtt{B}(v)$,
645 for all $v \in V(D)$. Let Υ_{slots} be the lexicographic ordering of slots. Let $i \in [n]$. We define the
646 set S_i as follows.

647 $$S_i = \{v \in V(D) \mid (\mathtt{pos}(\sigma(v)), \mathtt{bucket}(\sigma(v))) \text{ belongs to the first } i \text{ elements in } \Upsilon_{slots}\}$$

Algorithm 2 Algorithm for computing a collection of $(b+1)$-bucketing of a weakly connected digraph

Input: A weakly connected digraph D.
Output: A collection of $(b+1)$-bucketings of D.

1: Let v be an arbitrary vertex of D.
2: $\mathcal{B} = \{\mathtt{B}\colon V(D) \to [\lceil |V(D)|/(b+1) \rceil] \cup \{\star\} \mid \mathtt{B}(v) \in [\lceil |V(D)|/(b+1) \rceil]$ and $\mathtt{B}(w) = \star, \forall w \in V(D) \setminus \{v\}\}$. ▷ $\mathcal{B}$ is a collection of all possible $(b+1)$-bucketings for v.
3: $queue = (v)$
4: **while** $queue \neq \emptyset$ **do**
5: Let q be the first element of $queue$.
6: **for** each vertex $w \in N^+(q) \cup N^-(q)$ **do**
7: $\mathcal{B}' = \emptyset$. ▷ Initialize an empty collection of $(b+1)$-bucketings.
8: **for** each $\mathtt{B} \in \mathcal{B}$ **do**
9: $\mathtt{B}' = \mathtt{B}$ ▷ Initialize a $(b+1)$-bucketing as $\mathtt{B}$
10: $\mathtt{B}'(w) = \mathtt{B}(q)$ ▷ place w in the same bucket as that of q
11: $\mathcal{B}' = \mathcal{B}' \cup \{\mathtt{B}'\}$ ▷ add $(b+1)$-bucketing $\mathtt{B}'$ in the collection $\mathcal{B}'$
12: $\mathtt{B}'' = \mathtt{B}$ ▷ Initialize a new $(b+1)$-bucketing $\mathtt{B}''$ as $\mathtt{B}$
13: **if** $w \in N^+(q)$ **then**
14: $\mathtt{B}''(w) = \mathtt{B}(q) + 1$ ▷ place w in the bucket succeeding the bucket of q.
15: **else**
16: $\mathtt{B}''(w) = \mathtt{B}(q) - 1$ ▷ place w in the bucket preceding the bucket of q.
17: $\mathcal{B}' = \mathcal{B}' \cup \{\mathtt{B}''\}$ ▷ add newly constructed bucket $\mathtt{B}''$ in the collection $\mathcal{B}'$
18: $queue = queue \cup (w)$
19: $\mathcal{B} = \mathcal{B}'$ ▷ reinitialize $\mathcal{B}$ as $\mathcal{B}'$
20: Remove q from $queue$ ▷ all the out(in)-neighbours of q are placed in the buckets
21: Remove all those bucketings from $\mathcal{B}$ that are not $(b+1)$-topological bucketing.
22: **return** $\mathcal{B}$

Clearly, $|S_i| = i$. We claim that S_i is a lexicographically embeddible subset. We first prove (C1). Consider an arc $(u, v) \in E(D)$ such that $u \in S_i$ and $v \notin S_i$. For the contradiction, suppose that $\mathtt{B}(u) \neq \mathtt{B}(v)$. Since $u \in S_i$ and $v \notin S_i$, by the definition of S_i, $\mathtt{pos}(\sigma(u)) \leq \mathtt{pos}(\sigma(v))$. Since $\mathtt{B}$ is a topological bucketing and $\mathtt{B}(u) \neq \mathtt{B}(v)$, $\mathtt{B}(u) = \mathtt{B}(v) - 1$. Moreover, since $\mathtt{B}$ is a $(b+1)$-bucketing, we obtained that $\sigma(v) - \sigma(u) \geq b + 1$, a contradiction to the fact that σ is a b-ordering. Now, we will prove (C2). Consider an arc $(u, v) \in E(D)$ such that $v \in S_i$ and $u \notin S_i$. For the contradiction, suppose that $\mathtt{B}(u) \neq \mathtt{B}(v) - 1$. Since σ is a b-topological ordering, and $\mathtt{B}$ is $(b+1)$-bucketing, we have that either $\mathtt{B}(u) = \mathtt{B}(v)$ or $\mathtt{B}(u) = \mathtt{B}(v) - 1$. Thus, $\mathtt{B}(u) = \mathtt{B}(v)$. Moreover, since $v \in S_i$ and $u \notin S_i$, by the definition of S_i, we infer that $\mathtt{pos}(\sigma(v)) < \mathtt{pos}(\sigma(u))$. Hence, $\sigma(v) < \sigma(u)$, a contradiction to the fact that σ is a topological ordering of D. Note that $\sigma|_{S_i}$ is a b-topological ordering that satisfies (C3) by the construction of S_i.

Now, suppose that there exists a les, $\emptyset \subset S_1 \subset \cdots \subset S_n = V$. We construct an ordering σ as follows. Let $v \in S_i \setminus S_{i-1}$. We set $\sigma(v) = k$, where $(\mathtt{pos}(k), \mathtt{bucket}(k))$ is at i^{th} position in Υ_{slots}. We claim that σ is a b-topological ordering of the digraph D such that for all $v \in V(D)$, $\mathtt{B}_\sigma(v) = \mathtt{B}(v)$. Since each S_i is a lexicographically embeddible set, if a vertex $v \in S_i \setminus S_{i-1}$, then $\mathtt{B}(v) = ((i-1) \mod n/(b+1)) + 1$. Therefore, $\mathtt{B}_\sigma(v) = \mathtt{B}(v)$, for all $v \in V(D)$ by the construction of σ. Now, we show that σ is a topological ordering of D. For the contradiction, suppose that $(u, v) \in E(D)$ such that $\sigma(u) = p$, $\sigma(v) = q$ and $q < p$. Since $q < p$, either

667 $\texttt{pos}(q) \leq \texttt{pos}(p)$ or $\texttt{bucket}(q) < \texttt{bucket}(p)$. Suppose that $\texttt{pos}(q) \leq \texttt{pos}(p)$. Then, there
668 exists a lexicographical embeddible set S such that $v \in S$ and $u \notin S$. Therefore, by condition
669 (C2), $\texttt{B}(u) = \texttt{B}(v) - 1$. Hence, $\texttt{B}_\sigma(u) < \texttt{B}_\sigma(v)$ as $\texttt{B}_\sigma = \texttt{B}$. Since $q < p$, $\texttt{B}_\sigma(v) \leq \texttt{B}_\sigma(u)$, a
670 contradiction. Now, suppose that $\texttt{pos}(q) > \texttt{pos}(p)$. Then, there exists a lexicographical
671 embeddible set S such that $u \in S$ and $v \notin S$. Then, by condition (C1), $\texttt{B}(u) = \texttt{B}(v)$. Since
672 $q < p$ and $\texttt{pos}(q) > \texttt{pos}(p)$, we have that $\texttt{bucket}(q) < \texttt{bucket}(p)$. Since by the definition of
673 $\texttt{B}$ and $\texttt{bucket}$, $\texttt{B}_\sigma(w) = \texttt{B}(w) = \texttt{bucket}(\sigma(w))$, for all $w \in V(D)$, we have that $\texttt{B}(v) < \texttt{B}(u)$,
674 a contradiction. Therefore, σ is a topological ordering. Now, we prove that σ is a b-ordering.
675 Let $(u, v) \in E(D)$. Since σ is a topological ordering of $V(D)$, $\sigma(u) < \sigma(v)$. If $\texttt{B}_\sigma(u) = \texttt{B}_\sigma(v)$,
676 then $\sigma(v) - \sigma(u) \leq b$. Now, suppose that $\texttt{B}_\sigma(u) \neq \texttt{B}_\sigma(v)$. Since σ is a topological ordering
677 and $\sigma(u) < \sigma(v)$, $\texttt{B}_\sigma(u) < \texttt{B}_\sigma(v)$. If $\texttt{pos}(u) > \texttt{pos}(v)$, then $\sigma(v) - \sigma(u) \leq b$. Now, suppose
678 that $\texttt{pos}(u) \leq \texttt{pos}(v)$. Then, there exists a lexicographical embeddible set S such that $u \in S$
679 and $v \notin S$. Since $\texttt{B}_\sigma = \texttt{B}$, we have that $\texttt{B}(u) < \texttt{B}(v)$. Moreover, since $(u, v) \in E(D)$, this
680 contradicts that S is a lexicographically embeddible set (condition (C1) does not hold).

681 C.4 Proof of Lemma 3.5

We find les using a dynamic programming algorithm. We define our dynamic programming table as follows. For each subset $S \subseteq V(D)$, $c[S]$ is 1, if S is a lexicographically embeddible set, otherwise 0. We compute $c[S]$ for all 2^n subsets of $V(D)$. Base case of our algorithm is $c[\emptyset] = 1$. Clearly, $\emptyset$ is vacuously a lexicographically embeddible set. Now, for recursive computation, let $S \subseteq V(D)$. We compute $c[S]$ using the following formula.

$$c[S] = \vee_{v \in S}(c[S \setminus \{v\}] \wedge \texttt{In}(v) \wedge \texttt{Out}(v) \wedge \texttt{SB}(v))$$

682 In the above formula, $\texttt{In}(v) = 1$ if for all the arcs $(v, u) \in E(D)$ such that $u \notin S$, $\texttt{B}(v) = \texttt{B}(u)$,
683 otherwise 0; $\texttt{Out}(v) = 1$ if for all the arcs $(u, v) \in E(D)$ such that $u \notin S$, $\texttt{B}(u) < \texttt{B}(v)$,
684 otherwise 0; and $\texttt{SB}(v) = 1$ if $\texttt{B}(v)$ is equal to $((|S| - 1) \mod {}^{n}/_{(b+1)}) + 1$, otherwise 0. Note
685 that $\texttt{In}(v)$ and $\texttt{Out}(v)$ indicates whether condition (C1) and condition (C2) are satisfied for
686 the incoming and outgoing arcs incident on v. Now, we prove that $c[S] = 1$ if and only if S
687 is a lexicographically embeddible set, where $S \subseteq V(D)$. In the forward direction, suppose
688 that $c[S] = 1$. This implies that there exists $v \in S$ such that $c[S \setminus \{v\}] = 1$, $\texttt{In}(v) = 1$,
689 $\texttt{Out}(v) = 1$ and $\texttt{SB}(v) = 1$. Note that by the definition of $\texttt{In}(v)$ and $\texttt{Out}(v)$ and the fact
690 that $S \setminus \{v\}$ is a lexicographically embeddible set, conditions (C1) and (C2) in Definition
691 3.3 are satisfied by S. Since $c[S \setminus \{v\}] = 1$, there exists an ordering, $\rho \colon S \setminus \{v\} \to [n]$
692 that respects lexicographic ordering of slots. We define a function $\gamma \colon S \to [n]$ such that
693 $\gamma(u) = \rho(u)$ for all $u \in S \setminus \{v\}$ and $\gamma(v) = k$, where $(\texttt{pos}(k), \texttt{bucket}(k))$ is $|S|^{th}$ element
694 in the lexicographic ordering of slots. Since $\texttt{bucket}(\rho(u)) = \texttt{B}(u)$, for all $u \in S \setminus \{v\}$ and
695 $\texttt{SB}(v) = 1$, we obtained that $\texttt{bucket}(\gamma(u)) = \texttt{B}(u)$, for all $u \in S$. Now, we argue that γ is
696 a topological ordering. Consider an arc $(x, y) \in E(D)$ such that $x, y \in S$. Suppose that
697 $x \in S \setminus \{v\}$. If $y \in S \setminus \{v\}$, then since ρ is a topological ordering, $\gamma(x) < \gamma(y)$. Suppose
698 that $y \notin S \setminus \{v\}$, that is, $y = v$. Since $S \setminus \{v\}$ is a lexicographically embedidddble set,
699 using condition (C1), $\texttt{B}(x) = \texttt{B}(v)$. Hence, by the construction of γ, $\gamma(x) < \gamma(v)$. Now,
700 suppose that $x \notin S \setminus \{v\}$, that is, $x = v$. This implies that $y \in S \setminus \{v\}$. Since $S \setminus \{v\}$
701 is a lexicographically embeddible set, $\texttt{B}(v) = \texttt{B}(y) - 1$ (condition (C2) in Definition 3.3).
702 Hence, γ is a topological ordering. Now, we prove that γ is a b-ordering. Consider an arc
703 $(x, y) \in E(D)$ such that $x, y \in S$. If $x, y \in S \setminus \{v\}$, then $|\gamma(x) - \gamma(y)| \leq b$ as ρ is a b-ordering.
704 Suppose that $x = v$. Since $y \in S \setminus \{v\}$ and $S \setminus \{v\}$ is a lexicographically embeddible set,
705 using condition (C2) in Definition 3.3, $\texttt{B}(v) = \texttt{B}(y) - 1$. Therefore, by the construction of γ,
706 $\texttt{pos}(\gamma(v)) > \texttt{pos}(\gamma(y))$. Hence, $|\gamma(x) - \gamma(y)| \leq b$. Now, suppose that $y = v$. Recall that we

are considering arc (x, v). Since $x \in S \setminus \{v\}$, using condition (C1) in Definition 3.3, we have that $\mathtt{B}(v) = \mathtt{B}(x)$. Since $\mathtt{B}$ is a $(b+1)$-bucketing, we obtained that $|\gamma(v) - \gamma(x)| \leq b$ Hence, S is a lexicographically embeddible set. This completes the proof in the forward direction. In the backward direction, suppose that S is a lexicographically embeddible set. Then, there exists an ordering $\rho : S \to [n]$ that respects lexicographic ordering of *slots*. Let v be the vertex in S for which $(\mathtt{pos}(\rho(v)), \mathtt{bucket}(\rho(v)))$ is largest among all the vertices in S. We claim that $c[S \setminus \{v\}] = 1$, $\mathtt{In}(v) = 1$, $\mathtt{Out(v)} = 1$ and $\mathtt{SB}(v) = 1$. Note that since S is a lexicographically embeddible set, $\mathtt{In}(v) = 1$, $\mathtt{Out(v)} = 1$ and $\mathtt{SB}(v) = 1$. Now, we prove that $S \setminus \{v\}$ is a lexicographically embeddible set. We first prove $(C1)$. Suppose that $(u, u') \in E(D)$ such that $u \in S \setminus \{v\}$ and $u' \in V(D) \setminus (S \setminus \{v\})$. For the contradiction, suppose that $\mathtt{B}(u) \neq \mathtt{B}(u')$. If $u' \neq v$, then since $u' \notin S$, this contradicts that S is a lexicographically embeddible set. Now, suppose that $u' = v$. Since $\mathtt{B}$ is a topological bucketing and $\mathtt{B}(u) \neq \mathtt{B}(v)$, $\mathtt{B}(u) < \mathtt{B}(v)$. By the choice of v, $(\mathtt{pos}(\rho(u)), \mathtt{bucket}(\rho(u))) < (\mathtt{pos}(\rho(v)), \mathtt{bucket}(\rho(v)))$. Therefore, $\mathtt{pos}(\rho(u)) \leq \mathtt{pos}(\rho(v))$. Hence, $\rho(v) - \rho(u) \geq b + 1$, a contradiction to that ρ is a b-ordering. Now, we prove condition (C2). Suppose that there exists an arc $(u, u') \in E(D)$ such that $u' \in S \setminus \{v\}$ and $u \in V(D) \setminus (S \setminus \{v\})$. For the contradiction suppose that $\mathtt{B}(u) \geq \mathtt{B}(u')$. Since $\mathtt{B}$ is topological bucketing, $\mathtt{B}(u) \leq \mathtt{B}(u')$. Therefore, $\mathtt{B}(u) = \mathtt{B}(u')$. If $u \neq v$, then this contradicts that S is a lexicographically embeddible set. Suppose that $u = v$. As argued above $\mathtt{pos}(\rho(u')) \leq \mathtt{pos}(\rho(v))$. Moreover, since $\mathtt{B}(v) = \mathtt{B}(u')$, we obtained that $\mathtt{pos}(\rho(u')) < \mathtt{pos}(\rho(v))$. Since, $(v, u') \in E(D)$, this contradicts that ρ is a topological ordering. Note that $\rho|_{S \setminus \{v\}}$ is a b-topological ordering that satisfies condition (C3). ◀

D Missing Proofs of Section 5

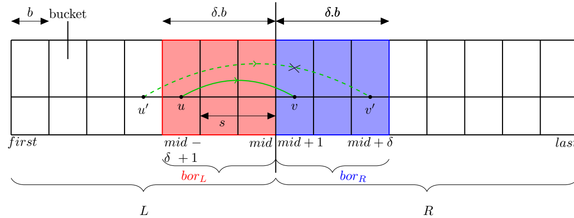

Figure 1 The buckets of Algorithm 1

▶ **Lemma D.1.** *Let $\mathcal{I} = (D, b, s, first, last, left\text{-}bor(V(D)), right\text{-}bor(V(D)), \mathtt{B}_{in})$ be a legitimate instance for Algorithm 1. If Algorithm 1 returns a bucketing, then it is a look-out bucketing for $\mathcal{I}$.*

Proof. Recall, from Line 1 of Algorithm 1 that $V = V(D)$ and $\delta = \lceil s+1/b \rceil$. Since $\mathcal{I}$ is a legitimate instance of Algorithm 1, $|V| = 2^{\eta} \cdot b \cdot \delta$, for some positive integer $\eta \geq 0$. We prove the lemma by induction on η. When $\eta = 0$, $|V| = b \cdot \delta$. In this case, since the total number of buckets in the instance $\mathcal{I}$ is $last - first + 1 = \delta$ (from (P2) of Definition 5.2), a b-bucketing of V always exists. Also, the external stretch of any b-bucketing is at most $(\delta - 2) \cdot b \leq s$. Thus, a look-out bucketing for $\mathcal{I}$ always exists.

Henceforth, $\eta > 1$. Observe from the pseudo-code of Algorithm 1 that, it returns a bucketing only at Line 3 and at Line 19. Since $\eta > 1$, $|V| > b \cdot \delta$, it remains to prove that the bucketing returned at Line 19 is a look-out bucketing for $\mathcal{I}$. Observe, again from the

741 pseudo-code of Algorithm 1, that a bucketing is returned at Line 19 only when the following
742 conditions hold.

743 **1.** At Line 5 there exists a partition, say (L, R) of V, such that $|L| = |R|$, $\mathtt{B}_{in}$ validates
744 (L, R).
745 **2.** During the run of the iteration of **for** loop at Line 5 for the (L, R) described above, at
746 Line 9 there exists $\mathtt{B} \in \mathcal{B}$ such that the instances $\mathcal{I}_L^{\mathtt{B}}$ and $\mathcal{I}_R^{\mathtt{B}}$ defined at Lines 15 and 16
747 are legitimate and, $Algorithm\ 1(\mathcal{I}_L^{\mathtt{B}})$! = NO and $Algorithm\ 1(\mathcal{I}_R^{\mathtt{B}})$! = NO.

748 Let $\mathtt{B}_{out}^L = Algorithm\ 1(\mathcal{I}_L^{\mathtt{B}})$ and $\mathtt{B}_{out}^R = Algorithm\ 1(\mathcal{I}_L^{\mathtt{B}})$. Since $|L| = |R|$ and $|V| = 2^\eta \cdot b \cdot \delta$,
749 we have that $|L| = |R| = 2^{\eta-1} \cdot b \cdot \delta$. Thus, from the induction hypothesis, $\mathtt{B}_{out}^L$ is a look-
750 out bucketing for $\mathcal{I}_L^{\mathtt{B}}$ and $\mathtt{B}_{out}^R$ is a look-out bucketing for $\mathcal{I}_R^{\mathtt{B}}$. We remain to show that
751 $\mathtt{B}_{out} = \mathtt{B}_{out}^L \cup \mathtt{B}_{out}^R$ is a look-out bucketing for $\mathcal{I}$. We prove this below.

752 **Bucketing:** Since $\mathtt{B}_{out}^L : L \to [first, mid]$ and $\mathtt{B}_{out}^R : R \to [mid+1, last]$ are b-bucketings, and
753 $L \cap R = \emptyset$, $L \cup R = V$, $[first, mid] \cap [mid+1, last] = \emptyset$ and $[first, mid] \cup [mid+1, last] =$
754 $[first, last]$, $\mathtt{B}_{out} : V \to [first, last]$ is a b-bucketing of V.
755 **Consistent:** We now prove that $\mathtt{B}_{out}$ is consistent with $\mathtt{B}_{in}$. Since $\mathcal{I}$ is a legitimate instance,
756 from (P4) of Definition 5.2, for each $v \in left\text{-}bor(V)$, $\mathtt{B}_{in}(v) \in [first, first + \delta - 1]$ and
757 for each $v \in right\text{-}bor(V)$, $\mathtt{B}_{in}(v) \in [last - \delta + 1, last]$. Since $\mathcal{I}_L^{\mathtt{B}}$ and $\mathcal{I}_R^{\mathtt{B}}$ are legitimate
758 instances, $left\text{-}bor(L) \subseteq L$ and $right\text{-}bor(R) \subseteq R$. Also, for each $v \in left\text{-}bor(L)$,
759 $\mathtt{B}_{in}^{newL}(v) \in [first, first + \delta - 1]$ and for each $v \in right\text{-}bor(R)$, $\mathtt{B}_{in}^{newR}(v) \in [last - \delta +$
760 $1, last]$. Since $\mathtt{B}_{out}^L$ (respectively $\mathtt{B}_{out}^R$) is a look-out bucketing for $\mathcal{I}_L^{\mathtt{B}}$ (respectively $\mathcal{I}_R^{\mathtt{B}}$), $\mathtt{B}_{out}^L$
761 (respectively $\mathtt{B}_{out}^R$) is consistent with $\mathtt{B}_{in}^{newL}$ (respectively $\mathtt{B}_{in}^{newR}$). Since, $\mathtt{B}_{out} = \mathtt{B}_{out}^L \cup \mathtt{B}_{out}^R$,
762 and $left\text{-}bor(L) = left\text{-}bor(V)$ and $right\text{-}bor(R) = right\text{-}bor(V)$ (from the construction
763 in Lines 13 and 14), we have that, for $v \in left\text{-}bor(V)$, $\mathtt{B}_{out}(v) \in [first, first + \delta - 1]$
764 and $v \in right\text{-}bor(V)$, $\mathtt{B}_{out}(v) \in [first + \delta - 1, last]$. Thus, $\mathtt{B}_{out}$ is consistent with $\mathtt{B}_{in}$.
765 **External Stretch:** Consider any arc $(u, v) \in E(D)$.

766 **1.** If $u, v \in L$ (respectively $u, v \in R$), then since $\mathtt{B}_{out} = \mathtt{B}_{out}^L \cup \mathtt{B}_{out}^R$ and the external
767 stretch of $\mathtt{B}_{out}^L$ (respectively $\mathtt{B}_{out}^R$) is at most s, we have that either $\mathtt{B}_{out}(u) \geq \mathtt{B}_{out}(v)$
768 or $(\mathtt{B}_{out}(v) - \mathtt{B}_{out}(u) - 1) \cdot b \leq s$.
769 **2.** If $u \in R$ and $v \in L$, then since $\mathtt{B}_{out} = \mathtt{B}_{out}^L \cup \mathtt{B}_{out}^R$, $\mathtt{B}_{out}(u) \geq \mathtt{B}_{out}(v)$.
770 **3.** If $u \in L$ and $v \in R$. In this case, first observe, from the construction of bor_L and
771 bor_R in Lines 6 and 7 of Algorithm 1, that $u \in bor_L$ and $v \in bor_R$. Since $\mathtt{B} \in \mathcal{B}$ (from
772 Line 9), from Line 8, $\mathtt{B}$ is a partial b-bucketing of $bor_L \cup bor_R$ of external stretch at
773 most s and $\mathtt{B}(u) < \mathtt{B}(v)$, therefore, $(\mathtt{B}(v) - \mathtt{B}(u) - 1) \cdot b \leq s$. Since $\mathtt{B}$ is consistent with
774 $\mathtt{B}_{in}$ (from Line 8), and $\mathtt{B}_{in}$ is consistent with $\mathtt{B}_{out}$ (as proved above), we conclude that
775 $(\mathtt{B}_{out}(v) - \mathtt{B}_{out}(u) - 1) \cdot b \leq s$.

776 ◀

777 ▶ **Lemma D.2.** *Let $\mathcal{I} = (D, b, s, first, last, left\text{-}bor(V(D)), right\text{-}bor(V(D)), \mathtt{B}_{in})$ be a legit-*
778 *imate instance for Algorithm 1 such that a look-out bucketing for $\mathcal{I}$ exists. Then, on input $\mathcal{I}$,*
779 *Algorithm 1 does not return* NO*.*

780 **Proof.** Recall that $V = V(D)$ and $\delta = \lceil s+1/b \rceil$. Since $\mathcal{I}$ is a legitimate instance, $|V| = 2^\eta \cdot b \cdot \delta$,
781 $\eta \geq 0$ is a positive integer. We prove this lemma by induction on η. When $\eta = 0$, Algorithm 1
782 always returns a look-out bucketing of $\mathcal{I}$ (Line 3). Henceforth, $\eta \geq 1$. From the pseudo-code
783 of Algorithm 1, to prove the lemma it is enough to prove the following.

784 **1.** There exists a partition of $V = (L, R)$, $|L| = |R|$ such that $\mathtt{B}_{in}$ validates (L, R),

2. The family $\mathcal{B}$ constructed at Line 8 (during the iteration of the **for** loop at Line 5 for (L, R)) contains a $\mathtt{B}$ such that the following holds.

(Q1) $\mathtt{B}$ is a partial b-bucketing of $bor_L \cup bor_R$, $\mathtt{B} : bor_L \cup bor_R \to [mid - \delta + 1, mid + \delta]$,

(Q2) For each $v \in bor_L$, $\mathtt{B}(v) \in [mid - \delta + 1, mid]$ and for each $v \in bor_R$, $\mathtt{B}(v) \in [mid + 1, mid + \delta]$,

(Q3) the external stretch of $\mathtt{B}$ is at most s,

(Q4) $\mathtt{B}$ is consistent with $\mathtt{B}_{in}$,

(Q5) The instances $\mathcal{I}_L^{\mathtt{B}}$ and $\mathcal{I}_R^{\mathtt{B}}$ defined at Lines 15 and 16 are legitimate, and

(Q6) *Algorithm* $1(\mathcal{I}_L^{\mathtt{B}})$ $! =$ NO, and *Algorithm* $1(\mathcal{I}_R^{\mathtt{B}})$ $! =$ NO.

Let $\mathtt{B}_{opt} : V \to [first, last]$ be a look-out bucketing for $\mathcal{I}$. Let $mid = {}^{(first+last-1)}/_2$. Since $\mathcal{I}$ is a legitimate instance, from (P2) of Definition 5.2, $last - first + 1 = 2^{\eta} \cdot \delta$. Therefore, $mid = {}^{(first+last-1)}/_2 = first + 2^{\eta-1} \cdot \delta - 1$. Thus, mid is an integer. Let $L = \bigcup_{i \in [mid]} \mathtt{B}_{opt}^{-1}(i)$ and $R = \bigcup_{i \in [mid+1, last]} \mathtt{B}_{opt}^{-1}(i)$. Clearly, (L, R) is a partition of V. Also, $|L| = |R|$ because $\mathtt{B}_{opt}$ is a b-bucketing of V and $|V|$ is divisible by b.

▷ Claim D.3. $\mathtt{B}_{in}$ validates (L, R).

Proof. Since $\mathcal{I}$ is a legitimate instance, from (P4) of Definition 5.2, for each $v \in \textit{left-bor}(V)$, $\mathtt{B}_{in}(v) \in [first, first + \delta - 1]$ and for each $v \in \textit{right-bor}(V)$, $\mathtt{B}_{in}(v) \in [last - \delta + 1, last]$.

From (P2) of Definition 5.2, $last - first + 1 = 2^{\eta} \cdot \delta$. Since $\eta \geq 1$ and $mid = {}^{(first+last-1)}/_2$, we have that $first + \delta - 1 \leq mid < last - \delta + 1$. Since $\mathtt{B}_{opt}$ is consistent with $\mathtt{B}_{in}$, from the description of $\mathtt{B}_{in}$ in (P4) of Definition 5.2, and from the construction of L and R, we have that, for each $v \in \textit{left-bor}(V)$, $v \in L$ and for each $v \in \textit{right-bor}(V)$, $v \in R$. Thus, $\mathtt{B}_{in}$ validates (L, R). ◁

Consider the execution of Lines 6 to 8 during the iteration of the **for** loop for the (L, R) constructed above. Consider the sets $bor_L = \{v \in L \mid$ there exists $u \in R$ such that $(v, u) \in E(D)\}$ and $bor_R = \{v \in R \mid$ there exists $u \in L$ such that $(u, v) \in E(D)\}$.

▷ Claim D.4. For each $v \in bor_L$, $mid - \delta + 1 \leq \mathtt{B}_{opt}(v) \leq mid$, and for each $v \in bor_R$, $mid + 1 \leq \mathtt{B}_{opt}(v) \leq mid + \delta$.

Proof. From the construction of L and R and the fact that $bor_L \subseteq L$ and $bor_R \subseteq R$, we have that, for each $v \in bor_L$, $\mathtt{B}_{opt}(v) \leq mid$, and for each $v \in bor_R$, $\mathtt{B}_{opt}(v) \geq mid + 1$. We now prove that for each $v \in bor_L$, $\mathtt{B}_{opt}(v) \geq mid - \delta + 1$. The proof of $\mathtt{B}_{opt}(v) \leq mid + \delta$ for $v \in bor_R$ follows using symmetric arguments.

Let $\mathtt{B}_{opt}(v) = i$. Suppose, for the sake of contradiction, that $i \leq mid - \delta$. Since $v \in bor_L$, there exists $u \in R$ such that $(v, u) \in E(D)$. Let $\mathtt{B}_{opt}(u) = j$. Since $u \in R$, $j \geq mid + 1$. Since the external stretch of $\mathtt{B}_{opt}$ is at most s and $\mathtt{B}_{opt}(u) < \mathtt{B}_{opt}(v)$, we have that $(j - i - 1) \cdot b \leq s$. Since $i \leq mid - \delta$ and $j \geq mid + 1$, we have that $(j - i - 1) \geq \delta$. Thus, we have that $\delta \cdot b \leq s$, that is $\lceil {}^{s+1}/_b \rceil \cdot b \leq s$, which is a contradiction. ◁

Define a partial b-bucketing $\mathtt{B} : bor_L \cup bor_R \to [mid - \delta + 1, mid + \delta]$ as follows: for each $v \in bor_L \cup bor_R$, $\mathtt{B}(v) = \mathtt{B}_{opt}(v)$. Then Claim D.4 proves (Q1) and (Q2). Since, the stretch of $\mathtt{B}_{opt}$ is at most s and $\mathtt{B}_{opt}$ is consistent with $\mathtt{B}_{in}$, from the construction of $\mathtt{B}$, (Q3) and (Q4) follow. Consider the execution of the **for** loop at Line 9 for the partial bucketing $\mathtt{B}$ constructed above. Let $\mathcal{I}_L^{\mathtt{B}}$ and $\mathcal{I}_R^{\mathtt{B}}$ be the instances as constructed at Lines 15 and 16 respectively.

▷ Claim D.5. $\mathcal{I}_L^{\mathtt{B}}$ and $\mathcal{I}_R^{\mathtt{B}}$ are legitimate instances of Algorithm 1.

828 Proof. We will prove the claim for $\mathcal{I}_L^{\mathtt{B}}$. The claim for $\mathcal{I}_R^{\mathtt{B}}$ follows from symmetric arguments.
829 Recall that $\mathcal{I}_L^{\mathtt{B}} = (D[L], b, s, first, mid, left\text{-}bor(L), right\text{-}bor(L), \mathtt{B}_{in}^{newL})$. The following
830 points prove that $\mathcal{I}_L^{\mathtt{B}}$ is a legitimate instance.

831 **1.** Since $|V| = 2^\eta \cdot b \cdot \delta$, $\eta \geq 1$ and $|L| = {}^{|V|}/_2$, $|L| = 2^{\eta-1} \cdot b \cdot \delta$.
832 **2.** Since $\mathcal{I}$ is a legitimate instance, from (P2) of Definition 5.2, we have that $last - first + 1 =$
833 $2^\eta \cdot \delta$. Since, $mid = {}^{(first+last-1)}/_2$, we have that $mid - first + 1 = 2^{\eta-1} \cdot \delta$.
834 **3.** Recall the construction of $left\text{-}bor(L)$ and $right\text{-}bor(L)$ from Line 13: $left\text{-}bor(L) =$
835 $left\text{-}bor(V)$ and $right\text{-}bor(L) = bor_L$. Since $left\text{-}bor(V) \subseteq L$ because $\mathtt{B}_{in}$ validates (L, R),
836 and $bor_L \subseteq L$ by definition, we have that $left\text{-}bor(L), right\text{-}bor(L) \subseteq L$.
837 **4.** Recall, from Lines 10 and 11, that $\mathtt{B}_{in}^{newL} = \mathtt{B}_{in}^{new}{}_{|L}$, where $\mathtt{B}_{in}^{new} = \mathtt{B} \cup \mathtt{B}_{in}$. First note that,
838 since $\mathtt{B}$ is a function from $bor_L \cup bor_R$, $\mathtt{B}_{in}$ is a function from $left\text{-}bor(V) \cup right\text{-}bor(V)$
839 and $left\text{-}bor(L) = left\text{-}bor(V)$, $right\text{-}bor(L) = bor_L$, we have that $\mathtt{B}_{in}^{newL}$ is a function
840 from $left\text{-}bor(L) \cup right\text{-}bor(L)$. Since $\mathtt{B}$ is consistent with $\mathtt{B}_{opt}$ (from the construction of
841 $\mathtt{B}$) and $\mathtt{B}_{in}$ is consistent with $\mathtt{B}_{opt}$ (because $\mathtt{B}_{opt}$ is a look-out bucketing for $\mathcal{I}$), we have
842 that $\mathtt{B}_{in}^{new} = \mathtt{B} \cup \mathtt{B}_{in}$ is consistent with $\mathtt{B}_{opt}$. Thus, since $\mathtt{B}_{opt}$ is a b-bucketing of V and has
843 external stretch at most s, we have that the $\mathtt{B}_{in}^{newL}$ is a partial b-bucketing and has external
844 stretch at most s. Also, since $\mathtt{B}_{in}^{newL}$ is consistent with $\mathtt{B}_{in}$ and $left\text{-}bor(L) = left\text{-}bor(V)$,
845 we have that for each $v \in left\text{-}bor(L)$, $\mathtt{B}_{in}^{newL}(v) \in [first, first + \delta - 1]$. Similarly,
846 since $\mathtt{B}_{in}^{newL}$ is consistent with $\mathtt{B}$, $right\text{-}bor(L) = bor_L$ and for each $v \in bor_L$, $\mathtt{B}(v) \in$
847 $[mid - \delta + 1, mid]$, we have that, for each $v \in right\text{-}bor(L)$, $\mathtt{B}_{in}^{newL}(v) \in [mid - \delta + 1, mid]$.
848 This proves that $\mathtt{B}_{in}^{newL}$ satisfies (P4) of Definition 5.2.

849 ◁

850 ▷ Claim D.6. *Algorithm* 1 $(\mathcal{I}_L^{\mathtt{B}})$! = NO and *Algorithm* 1$(\mathcal{I}_R^{\mathtt{B}})$! = NO.

851 Proof. We will prove the claim for $\mathcal{I}_L^{\mathtt{B}}$. The claim for $\mathcal{I}_R^{\mathtt{B}}$ follows from symmetric arguments.
852 We will now show that $\mathtt{B}_{opt|L}$ is a look-out bucketing of $\mathcal{I}_L^{\mathtt{B}}$. The proof of the claim then
853 follows from induction hypothesis.
854 Since $\mathcal{B}_{opt}$ is a b-bucketing, so is $\mathtt{B}_{opt|L}$. From the construction of L, $\mathtt{B}_{opt|L} : L \to$
855 $[first, mid]$. Since $\mathtt{B}_{in}^{newL} = \mathtt{B}_{in}^{new}{}_{|L}$, where $\mathtt{B}_{in}^{new} = \mathtt{B} \cup \mathtt{B}_{in}$, and $\mathtt{B}_{out}$ is consistent with $\mathtt{B}_{in}$
856 and $\mathtt{B}$, we conclude that $\mathtt{B}_{opt|L}$ is consistent with $\mathtt{B}_{in}^{newL}$. Since the external stretch of $\mathtt{B}_{opt}$ is
857 at most s, so is the external stretch of $\mathtt{B}_{opt|L}$. ◁

858 Claims D.5 and D.6 prove (Q5) and (Q6). This finishes the proof of the lemma. ◀

859 ▶ **Lemma D.7.** *On a legitimate input* $(D, b, s, first, last, left\text{-}bor(V(D)), right\text{-}bor(V(D)),$
860 $\mathtt{B}_{in})$*, Algorithm 1 returns a look-out bucketing for* $\mathcal{I}$*, if it exists.*

861 **Proof.** The proof follows from Lemma D.1 and D.2. ◀

862 ▶ **Lemma D.8.** *Algorithm 1 on a legitimate input* $(D, b, s, first, last, left\text{-}bor(V), right\text{-}bor(V),$
863 $B_{in})$*, runs in time* $\min\{\mathcal{O}^*(4^n \cdot \lceil {}^{s+1}/_b \rceil^n), \mathcal{O}^*(4^n \cdot \lceil {}^{s+1}/_b \rceil^{2(b+s)\log n})\}$*, where* n *is the number*
864 *of vertices of* D*.*

865 **Proof.** Recall that $\delta = \lceil {}^{s+1}/_b \rceil$. We begin by analysing the size and the time taken to compute
866 the family $\mathcal{B}$ at Line 8 for a fixed partition (L, R) of V. For this, we can safely assume that
867 $|V| = 2^\eta \cdot b \cdot \delta$ where $\eta \geq 1$. Recall that $\mathtt{B}_{in} : left\text{-}bor(V) \cup right\text{-}bor(V) \to [first, last]$. Let
868 $b_L = |bor_L \setminus left\text{-}bor(V)|$ and let $b_R = |bor_R \setminus right\text{-}bor(V)|$. Observe that the number of
869 functions from $bor_L \cup bor_R$ that map each vertex of bor_L in the range $[mid - \delta + 1, mid]$ and

map each vertex of bor_R in the range $[mid+1, mid+\delta]$ and are consistent with $\mathtt{B}_{in}$ is $\delta^{b_L} \cdot \delta^{b_R}$. Thus, the size of the family $\mathcal{B}$ at the iteration of the **for** loop at Line 5 corresponding to (L, R) is at $\delta^{b_L} \cdot \delta^{b_R}$. Also, the time taken to compute such a family is $\mathcal{O}(\delta^{b_L} \cdot \delta^{b_R})$.

Since $\mathcal{I}$ is a legitimate instance, from (P2) in Definition 5.2, $last - first + 1 = 2^{\eta} \cdot b \cdot \delta$. Since $\eta \geq 1$, we have that $first + \delta - 1 < last - \delta + 1$. Thus, from (P4) of Definition 5.2, $left\text{-}bor(V) \cap right\text{-}bor(V) = \emptyset$.

Again, for a fixed partition (L, R) of V considered during an iteration of the **for** loop at Line 5, consider the instances $\mathcal{I}_L^{\mathtt{B}}$ and $\mathcal{I}_R^{B}$ constructed at Lines 15 and 16, for any $\mathtt{B} \in \mathcal{B}$ at Line 9. Let $\mathcal{I}$ be the input instance. Let us denote by u the number of vertices that are not in $left\text{-}bor(V) \cup right\text{-}bor(V)$, that is, $u = |V(D) \setminus (left\text{-}bor(V) \cup right\text{-}bor(V))|$. We call these vertices the unplaced vertices for $\mathcal{I}$. Similarly, let u_L (respectively u_R) denote the number of unplaced vertices for $\mathcal{I}_L^{\mathtt{B}}$ (respectively $\mathcal{I}_R^{\mathtt{B}}$), that is $u_L = |L \setminus (left\text{-}bor(L) \cup right\text{-}bor(L))|$ (respectively $u_R = |R \setminus (left\text{-}bor(R) \cup right\text{-}bor(R))|$).

▷ Claim D.9. $u = u_L + u_R + b_L + b_R$.

Proof. For ease of notation, let $x_1 = |left\text{-}bor(V)|$ and let $x_2 = |right\text{-}bor(V)|$. Since $left\text{-}bor(V) \cap right\text{-}bor(V) = \emptyset$, we have that $u = |V| - (x_1 + x_2)$. Also, $u_L = |L| - (b_L + x_1)$ and $u_R = |R| - (b_R + x_2)$. Since $|V| = |L| + |R|$ (because $|V|$ is a power of 2 and (L, R) is a partition of V), we have that $u = u_L + u_R + b_L + b_R$. ◁

Running time analysis 1: We are now equipped to analyse the running time of Algorithm 1. Let $T(n, u)$ denote the time taken by Algorithm 1 on input $\mathcal{I}$, where the number of vertices in the digraph in the instance is n and the number of unplaced vertices is u. In the following, c is some fixed constant. From Lines 2 to 3, we have that $T(b \cdot \delta) \leq c$. Since the time taken to execute Lines 6 to 8 is at most $\mathcal{O}(|\mathcal{B}| + n) \leq c \cdot (|\mathcal{B}| + n)$, and for a fixed partition (L, R), $|\mathcal{B}| \leq \delta^{b_L + b_R}$, we have the following recurrence.

$$T(n, u) \leq \sum_{\substack{V=(L,R),\\ |L|=|R|}} c \cdot (\delta^{b_L + b_R} + n) + \delta^{b_L + b_R}(T(n/2, u_L) + T(n/2, u_R)) \quad (1)$$

By induction on n, we can prove that for any $u \leq n$, $T(n, u) = \mathcal{O}^*(4^n \cdot \delta^n)$.

Running time analysis 2: One can do the running time analysis of the algorithm a little differently to get a different running time. The crucial point of this analysis is the fact that in each iteration of the **for** loop at Line 5, if the set constructed at Line 8 is not empty, that is the **for** loop at Line 9 is executed, then the sizes of the sets bor_L and bor_R are bounded. This is formalized below.

▷ Claim D.10. Consider any arbitrary iteration of the **for** loop at Line 5. Let (L, R) be the partition considered during this iteration. Then if $\mathcal{B} \neq \emptyset$, then $|bor_L|, |bor_R| \leq b + s$.

Proof. For the sake of contradiction, suppose that $|bor_L| > b + s - 1$ (symmetric arguments hold for bounding the size of bor_R). Since $\mathcal{B} \neq \emptyset$, there exists a b-bucketing $\mathtt{B} : bor_L \cup bor_R \to [mid - \delta + 1, mid + \delta]$ such that for each $v \in bor_L$, $\mathtt{B}(v) \in [mid - \delta + 1, mid]$ and for each $v \in bor_R$, $\mathtt{B}(v) \in [mid + 1, mid + \delta]$. Let $v \in bor_L$ be the vertex such that $\mathtt{B}(v)$ is minimum. Let $\mathtt{B} = i$. Then all the vertices of bor_L appear in the buckets numbered i to mid. Since $v \in bor_L$, there exists a vertex $u \in R$ such that $(v, u) \in E(D)$, and the external stretch of $\mathtt{B}$ is at most s, the number of vertices that appear in the buckets numbered from $i + 1$ to mid is at most s. Since the size of each bucket is b, the total number of vertices that appear in the buckets numbered between i and mid is at most $b + s$. Thus, the size of the set bor_L is at most $b + s$. ◁

Since an alternate (and easier) bound on the size of the family $\mathcal{B}$ constructed at Line 8 is $\delta^{|bor_L|+|bor_R|}$, from Claim D.10, we conclude that $|| \leq \delta^{2(b+s)}$. Let $T(n)$ denote the time taken by the algorithm to solve an instance $\mathcal{I}$ where the digraph has n vertices. Let c be some fixed constant. Then, we have the following recurrence.

$$T(n) \leq c \cdot \delta^{2(b+s)} \cdot 2T(n/2) + cn \tag{2}$$

By induction on n, one can prove that $T(n) = \mathcal{O}^*(4^n \cdot \delta^{2(b+s)\cdot \log n})$. ◀

▶ **Lemma 5.5.** ($\star$) Algorithm 1 on a legitimate input $(D, b, s, first, last, left\text{-}bor(V), right\text{-}bor(V), B_{in})$, runs in time $\min\{\mathcal{O}^*(4^n \cdot \lceil s+1/b \rceil^n), \mathcal{O}^*(4^n \cdot \lceil s+1/b \rceil^{2(b+s)\log n})\}$, and returns a look-out bucketing for $\mathcal{I}$, if it exists.

Proof. The proof of the lemma follows from Lemmas D.7 and D.8. ◀

D.1 Proofs of Lemma 5.1 and Theorems 1.3 and 1.4

Proof of Lemma 5.1. Given a digraph D and positive integers b, s, create an instance $\mathcal{I} = (D, b, s, 1, |V|/b, \emptyset, \emptyset, \phi)$. Note that if $|V| = 2^\eta \cdot b \cdot \delta$, where $\delta = \lceil s+1/b \rceil$ and $\eta \geq 0$, then clearly $\mathcal{I}$ is a legitimate instance of Algorithm 1 and the proof of the lemma follows from Lemmas D.7 and D.8. In the other case, when the number of vertices is not as described above, then it is not difficult to see that an additional set of isolated vertices (at most the same number as the original number of vertices) can be added to the digraph without changing the solution or the running time of Algorithm 1 (beyond a multiplicative factor of 2). ◀

Lemma 5.1 forms the basis of the proofs of both Theorem 1.4 and Theorem 1.3. We give the proofs of these theorems below.

Proof of Theorem 1.4:

Let (D, b) be an instance of Digraph Bandwidth. Then run the algorithm of Lemma 5.1 on the instance where the digraph is D, the bucket size is 1 and the external stretch is $b-1$. The algorithm of Lemma 5.1 then returns a 1-bucketing of $V(D)$ of external stretch at most $b-1$, if it exists. Observe that such a bucketing of $V(D)$ is also an ordering of $V(D)$ of directed bandwidth at most b, and vice-versa. Thus, the proof of the theorem follows from Lemma 5.1.

We now ready to prove Theorem 1.3.

▶ **Lemma D.11.** *Let D be a digraph, b be a positive integer and $\epsilon > 0$ be a real. Then, there is an algorithm, that given (D, b, ϵ), runs in time $\mathcal{O}^*(4^n \cdot (\lceil 4/\epsilon \rceil)^n)$, and either concludes that the directed bandwidth of D is strictly greater than b, or outputs an ordering whose directed bandwidth is at most $(1+\epsilon) \cdot b$.*

Proof. If $b < 4/\epsilon$, run the algorithm of Theorem 1.4 on the instance (D, b). Note that it takes time $\mathcal{O}^* 4^n \cdot (\lceil 4/\epsilon \rceil)^n$. Henceforth, we are in the case when $b \geq 4/\epsilon$. Let $b' = \lfloor b\epsilon/2 \rfloor$. Then use the algorithm of Lemma 5.1 on the instance containing the digraph D, bucket size b' and external stretch $b-1$. Note that, $\lceil b/\lfloor b' \rfloor \rceil \leq \lceil 4/\epsilon \rceil$, whenever $b \geq \lceil 4/\epsilon \rceil$. We now prove that if D has directed bandwidth at most b, then the algorithm of Lemma 5.1 does not report NO.

▷ Claim D.12. If the directed bandwidth of D is at most b, then there is a b'-bucketing of $V(D)$ of external stretch at most $b-1$.

953 Proof. Let σ be an ordering of D of directed bandwidth at most b. Consider the unique
954 b'-bucketing induced by σ, say $\mathtt{B}$. We show that the external stretch of $\mathtt{B}$ is at most $b-1$.
955 For this, consider any arc $(u,v) \in E(D)$ such that $\mathtt{B}(u) < \mathtt{B}(v)$. Then, $\sigma(u) < \sigma(v)$. Let
956 $\mathtt{B}(u) = i$ and $\mathtt{B}(v) = j$. For the sake of contradiction, suppose that $(j-i-1)\cdot b' \geq b$. From
957 the construction of $\mathtt{B}$, we have that $\sigma(u) \in [(i-1)\cdot b', i \cdot b]$ and $\sigma(v) \in [(j-1)\cdot b', j\cdot b]$. Thus,
958 $\sigma(v) - \sigma(u) - 1 \geq (j-i-1)\cdot b' \geq b$, that is, $\sigma(v)-\sigma(u) \geq b+1$, which contradicts that the
959 directed bandwidth of σ is at most b. ◁

960 Thus, from Claim D.12, we conclude that, if the algorithm of Lemma 5.1 reports NO,
961 then the directed bandwidth of D is strictly greater than b. Otherwise, let $\mathtt{B}_{out}$ be the
962 b-bucketing outputted by the algorithm of Lemma 5.1. For each $i \in [1, \mathtt{B}|V|b']$, let σ^i be
963 some ordering of $\mathtt{B}^{-1}_{out}(i)$. Let $\sigma = \sigma^1 \cdot \ldots \cdot \sigma^{ns/b}$.

964 ▷ Claim D.13. The bandwidth of σ is at most $(1+\epsilon)\cdot b$.

965 Proof. Consider any arc $(u,v) \in E(D)$ such that $\sigma(u) < \sigma(v)$. Let $\mathtt{B}_{out}(u) = i$ and $\mathtt{B}_{out}(v) = j$.
966 Then $\sigma(v) - \sigma(u) \leq (j-i-1)\cdot b' + |\mathtt{B}^{-1}_{out}(i)| + |\mathtt{B}^{-1}_{out}(i)|$. Since, the external stretch of $\mathtt{B}$ is at
967 most $b-1$, we have that $(j-i-1)\cdot b' \leq b$. Thus, $\sigma(v)-\sigma(u) \leq b + 2\cdot b' = b + 2\cdot \lfloor b\cdot\epsilon/2 \rfloor \leq$
968 $b + b\cdot\epsilon = (1+\epsilon)\cdot b$. ◁

969 This proves the lemma. ◀

970 **Proof of Theorem 1.3**

971 Recall D is the input digraph and $\epsilon > 0$ is a real. We initial $b = 0$. Then run the algorithm of
972 Lemma D.11 on the instance (D, b, ϵ). If the algorithm outputs an ordering then we output
973 the same ordering, otherwise we increment b by 1 are repeat this procedure. Let OPT denote
974 the directed bandwidth of D. Then, for some $b \leq OPT$, algorithm of Lemma D.11 returns
975 an ordering whose bandwidth is at most $(1+\epsilon)\cdot b \leq (1+\epsilon)\cdot OPT$. This proves the theorem.

976
977 ## E Exact Algorithm for Digraph Bandwidth via Directed Homomorphisms

978 The goal of this section is to prove Theorem 1.5. Towards this, we give a reduction from
979 Digraph Bandwidth to a subgraph homomorphism problem for digraphs. Given two
980 digraphs D and H, a *directed homomorphism from D to H* is a function $h : V(D) \to V(H)$,
981 such that if $(u,v) \in E(D)$, then $(h(u), h(v)) \in E(H)$. A directed homomorphism that is
982 injective is called an *injective directed homomorphism*. For any positive integers n, b such
983 that $b \leq n$, we denote by P^b_n the directed graph on n vertices such that $V(P^b_n) = [n]$ and
984 $E(P^b_n) = E_f \uplus E_b$, where $E_b = \{(i,j) : i > j, i,j \in [n]\}$ and $E_f = \{(i, i+j) : i \in [n-1], j \in [b]\}$.
985 In the following lemma, we build the relation between Digraph Bandwidth of D and
986 injective homomorphism from D to P^b_n.

987 ▶ **Lemma E.1.** *For any digraph D and an integer b, D has bandwidth at most b if and only*
988 *if there is an injective homomorphism from D to P^b_n.*

989 **Proof.** In the forward direction, suppose that D has digraph bandwidth at most b. Let σ be
990 a b-ordering of D. Let $f \colon V(D) \to V(P^b_n)$ be a function such that $f(u) = \sigma(u)$. We claim
991 that f is an injective homomorphism. Since σ is an ordering of D, f is an injective function.
992 We prove that it is also a homomorphism. Consider an arc $(u,v) \in E(D)$. If $\sigma(u) < \sigma(v)$,
993 then since σ is a b-ordering, $\sigma(v) - \sigma(u) \leq b$. Therefore, $(f(u), f(v)) \in E_f$. If $\sigma(u) > \sigma(v)$,

994 then $(f(u), f(v)) \in E_b$. Hence, $(f(u), f(v)) \in E(P_n^b)$. In the backward direction, suppose
995 that there exists an injective homomorphism from D to P_n^b. Let $f : V(D) \to V(P_n^b)$ be
996 a function. Then, we claim that $\sigma = (f^{-1}(1), \cdots, f^{-1}(n))$ is a b-ordering of D. Suppose
997 not, then there exists an arc $(u, v) \in E(D)$ such that $\sigma(v) - \sigma(u) > b$. Let $u = f^{-1}(j)$
998 and $v = f^{-1}(k)$. Note that $\sigma(u) = j$ and $\sigma(v) = k$. Therefore, $j < k$. Since $k - j > b$,
999 $(j, k) \notin E(P_n^b)$, a contradiction that f is an injective homomorphism. ◀

1000 For any two digraphs D and H, let $inj(D, H)$ denote the number of injective homomor-
1001 phisms from D to H and let $hom(D, H)$ denote the number of homomorphisms from D to
1002 H. Then the following lemma holds from Theorem 1 in [1].

1003 ▶ **Lemma E.2** (Theorem 1, [1]). *Suppose that D and H be two digraphs such that $|V(D)| =$*
1004 *$|V(H)|$. Then,*

$$inj(D, H) = \sum_{W \subseteq V(D)} (-1)^{|W|} hom(D \setminus W, H).$$

1005 Now, we state the following known result about the number of homomorphisms between two
1006 given digraphs D and H.

1007 ▶ **Lemma E.3** (Theorem 3.1, 5.1 [11]). *Given digraphs D and H together with a tree*
1008 *decomposition of D of width $\mathtt{tw}$, $hom(D, H)$ can be computed in time $\mathcal{O}(nh^{\mathtt{tw}+1} \min\{\mathtt{tw}, h\})$,*
1009 *where $n = |V(D)|$ and $h = |V(H)|$.*

1010 Using Lemmas E.2 and E.3, we get the following result.

1011 ▶ **Lemma E.4.** *Given digraphs D and H together with a tree decomposition of D of width*
1012 *$\mathtt{tw}$, $inj(D, H)$ can be computed in time $\mathcal{O}(2^n nh^{\mathtt{tw}+1} \min\{\mathtt{tw}, h\})$, where $n = |V(D)|$ and*
1013 *$h = |V(H)|$.*

1014 **Proof of Theorem 1.5.** The proof follows from Lemmas E.1 and E.4. ◀