Adjacency Labelling for Planar Graphs (and Beyond)

Vida Dujmović
School of Computer Science and Electrical Engineering
University of Ottawa
Ottawa, Canada
vdujmovi@uottawa.ca

Louis Esperet
Laboratoire G-SCOP
CNRS, Univ. Grenoble Alpes
Grenoble, France
louis.esperet@grenoble-inp.fr

Cyril Gavoille
LaBRI
University of Bordeaux
Bordeaux, France
gavoille@labri.fr

Gwenaël Joret
Département d'Informatique
Université Libre de Bruxelles
Brussels, Belgium
gjoret@ulb.ac.be

Piotr Micek
Theoretical Computer Science Department
Faculty of Math. and Comp. Sci.
Jagiellonian University
Kraków, Poland
piotr.micek@gmail.com

Pat Morin
School of Computer Science
Carleton University
Ottawa, Canada
morin@scs.carleton.ca

***Abstract*—We show that there exists an adjacency labelling scheme for planar graphs where each vertex of an n-vertex planar graph G is assigned a $(1+o(1))\log_2 n$-bit label and the labels of two vertices u and v are sufficient to determine if uv is an edge of G. This is optimal up to the lower order term and is the first such asymptotically optimal result. An alternative, but equivalent, interpretation of this result is that, for every positive integer n, there exists a graph U_n with $n^{1+o(1)}$ vertices such that every n-vertex planar graph is an induced subgraph of U_n. These results generalize to a number of other graph classes, including bounded genus graphs, apex-minor-free graphs, bounded-degree graphs from minor closed families, and k-planar graphs.**

I. Introduction

A family $\mathcal{G}$ of graphs has an *$f(n)$-bit adjacency labelling scheme* if there exists a function $A\colon (\{0,1\}^*)^2 \to \{0,1\}$ such that for every n-vertex graph $G \in \mathcal{G}$ there exists $\ell\colon V(G) \to \{0,1\}^*$ such that $|\ell(v)| \leqslant f(n)$ for each vertex v of G and such that, for every two vertices v, w of G,

$$A(\ell(v),\ell(w)) = \begin{cases} 0 & \text{if } vw \notin E(G);\\ 1 & \text{if } vw \in E(G).\end{cases}$$

Let $\log x := \log_2 x$ denote the binary logarithm of x throughout the paper. In this paper we prove the following result:

The full version of the paper is available on arXiv at `https://arxiv.org/abs/2003.04280`. The research of V. Dujmović and P. Morin. was partially supported by NSERC. L. Esperet was partially supported by the French ANR Projects ANR-16-CE40-0009-01 (GATO) and ANR-18-CE40-0032 (GrR). C. Gavoille was partially supported by the French ANR projects ANR-16-CE40-0023 (DESCARTES) and ANR-17-CE40-0015 (DISTANCIA). G. Joret was supported by an ARC grant from the Wallonia-Brussels Federation of Belgium and by a grant from the National Fund for Scientific Research (FNRS). P. Micek was partially supported by the Polish National Science Center grant (BEETHOVEN; UMO-2018/31/G/ST1/03718).

Theorem 1. *The family of planar graphs has a $(1+o(1))\log n$-bit adjacency labelling scheme.*

Theorem 1 is optimal up to the lower order term, which is $\mathcal{O}\left(\sqrt{\log n \log\log n}\right)$ in our proof. An alternative, but equivalent, interpretation of Theorem 1 is that, for every integer $n \geqslant 1$, there exists a graph U_n with $n^{1+o(1)}$ vertices such that every n-vertex planar graph is isomorphic to some vertex-induced subgraph of U_n.[1]

Note that the proof of Theorem 1 is constructive: it gives an algorithm producing the labels in $\mathcal{O}(n \log n)$ time.

A. Previous Work

The current paper is the latest in a series of results dating back to Kannan, Naor, and Rudich [19], [20] and Muller [24] who defined adjacency labelling schemes[2] and described $\mathcal{O}(\log n)$-bit adjacency labelling schemes for several classes of graphs, including planar graphs. Since this initial work, adjacency labelling schemes and, more generally, informative labelling schemes have remained a very active area of research [2], [8], [1], [5], [7], [6], [9], [3].

Here we review results most relevant to the current work, namely results on planar graphs and their supporting results on trees and bounded-treewidth graphs. First, a superficial review: Planar graphs have been shown to have $(c+o(1))\log n$-bit adjacency labelling schemes for successive values of $c = 6, 4, 3, 2, \frac{4}{3}$ and finally Theorem 1

[1]There is a small technicality that the equivalence between adjacency labelling schemes and universal graphs requires that $\ell\colon V(G) \to \{0,1\}^*$ be injective. The labelling schemes we discuss satisfy this requirement. For more details about the connection between labelling schemes and universal graphs, the reader is directed to Spinrad's monograph [26, Section 2.1].

[2]There are some small technical differences between the two definitions that have to do with the complexity of computing $\ell(\cdot)$ as a function of G and $A(\cdot,\cdot)$.

2575-8454/20/$31.00 ©2020 IEEE
DOI 10.1109/FOCS46700.2020.00060

gives the optimal[3] result $c = 1$. We now give details of these results.

Muller's scheme for planar graphs [24] is based on the fact that planar graphs are 5-degenerate. This scheme orients the edges of the graph so that each vertex has 5 outgoing edges, assigns each vertex v an arbitrary $\lceil \log n \rceil$-bit identifier, and assigns a label to v consisting of v's identifier and the identifiers of the targets of v's outgoing edges. In this way, each vertex v is assigned a label of length at most $6\lceil \log n \rceil$. Kannan, Naor, and Rudich [20] use a similar approach that makes use of the fact that planar graphs have arboricity 3 (so their edges can be partitioned into three forests [25]) to devise an adjacency labelling scheme for planar graphs whose labels have length at most $4\lceil \log n \rceil$.

A number of $(1 + o(1)) \log n$-bit adjacency labelling schemes for forests have been devised [12], [9], [4], culminating with a recent $(\log n + \mathcal{O}(1))$-bit adjacency labelling scheme [4] for forests. Combined with the fact that planar graphs have arboricity 3, these schemes imply $(3 + o(1)) \log n$-bit adjacency labelling schemes for planar graphs.

A further improvement, also based on the idea of partitioning the edges of a planar graph into simpler graphs was obtained by Gavoille and Labourel [18]. Generalizing the results for forests, they describe a $(1 + o(1)) \log n$-bit adjacency labelling scheme for n-vertex graphs of bounded treewidth. As is well known, the edges of a planar graph can be partitioned into two sets, each of which induces a bounded treewidth graph. This results in a $(2 + o(1)) \log n$-bit adjacency labelling scheme for planar graphs.

Very recently, Bonamy, Gavoille, and Pilipczuk [10] described a $(4/3 + o(1)) \log n$-bit adjacency labelling scheme for planar graphs based on a recent *graph product structure theorem* of Dujmović et al. [15]. This product structure theorem states that any planar graph is a subgraph of a strong product $H \boxtimes P$ where H is a bounded-treewidth graph and P is a path. See Figure 1. It is helpful to think of $H \boxtimes P$ as a graph whose vertices can be partitioned into $h := |V(P)|$ *rows* $H_1, \ldots, H_h$, each of which induces a copy of H and with vertical and diagonal edges joining corresponding and adjacent vertices between consecutive rows.

The product structure theorem quickly leads to a $(1 + o(1)) \log(mh)$-bit labelling scheme where $m := |V(H)|$ and $h := |V(P)|$ by using a $(1 + o(1)) \log m$-bit labelling scheme for H (a bounded treewidth graph), a $\lceil \log h \rceil$-bit labelling scheme for P (a path), and a constant number of bits to locally encode the subgraph of $H \boxtimes P$ (of constant arboricity). However, for an n-vertex graph G that is a subgraph of $H \boxtimes P$ in the worst case m and h are each $\Omega(n)$, so this offers no immediate improvement over the existing $(2 + o(1)) \log n$-bit scheme.

[3]It is easy to see that, in any adjacency labelling scheme for any n-vertex graph G in which no two vertices have the same neighbourhood, all labels must be distinct, so some label must have length at least $\lceil \log n \rceil$.

Bonamy, Gavoille, and Pilipczuk improve upon this by cutting P (and hence G) into subpaths of length $n^{1/3}$ in such a way that this corresponds to removing $\mathcal{O}(n^{2/3})$ vertices of G that have a neighbourhood of size $\mathcal{O}(n^{2/3})$. The resulting (cut) graph is a subgraph of $H' \boxtimes P'$ where H' has bounded treewidth, $|H'| \leqslant n$, and P' is a path of length $n^{1/3}$ so it has a labelling scheme in which each vertex has a label of length $(1 + o(1)) \log(|H'| \cdot |P'|) \leqslant (4/3 + o(1)) \log n$. A slight modification of this scheme allows for the $\mathcal{O}(n^{2/3})$ *boundary* vertices adjacent to the cuts to have shorter labels, of length only $(2/3 + o(1)) \log n$. The cut vertices and the boundary vertices induce a bounded-treewidth graph of size $\mathcal{O}(n^{2/3})$. The vertices in this graph receive secondary labels of length $(2/3 + o(1)) \log n$. In this way, every vertex receives a label of length at most $(4/3 + o(1)) \log n$.

B. New Results

The adjacency labelling scheme described in the current paper is also based on the product structure theorem for planar graphs, but it avoids cutting the path P, and thus avoids boundary vertices that take part in two different labelling schemes. Instead, it uses a weighted labelling scheme on the rows $H_1, \ldots, H_h$ of $H \boxtimes P$ in which vertices that belong to H_i receive a label of length $(1 + o(1)) \log n - \log W_i$ where W_i is related to the number of vertices of G contained in H_i and H_{i-1}. The vertices of G in row i participate in a secondary labelling scheme for the subgraph of G contained in H_i and H_{i-1} and the labels in this scheme have length $\log W_i + o(\log n)$. Thus every vertex receives two labels, one of length $(1 + o(1)) \log n - \log W_i$ and another of length $\log W_i + o(\log n)$ for a total label length of $(1 + o(1)) \log n$.

The key new technique that allows all of this to work is that the labelling schemes of the rows $H_1, \ldots, H_h$ are not independent. All of these labelling schemes are based on a single balanced binary search tree T that undergoes insertions and deletions resulting in a sequence of related binary search trees $T_1, \ldots, T_h$ where each T_i represents all vertices of G in H_i and H_{i-1} and the label assigned to a vertex of H_i is essentially based on a path from the root of T_i to some vertex of T_i. By carefully maintaining the binary search tree T, the trees T_{i-1} and T_i are similar enough so that the label for v in H_i can be obtained, with $o(\log n)$ additional bits from the label for v in H_{i-1}.

The product structure theorem has been generalized to a number of additional graph families including bounded-genus graphs, apex-minor free graphs, bounded-degree graphs from minor-closed families, k-planar graphs, powers of bounded-degree bounded genus graphs, and k-nearest neighbour graphs of points in $\mathbb{R}^2$ [15], [16]. As a side-effect of designing a labelling scheme to work directly on subgraphs of a strong product $H \boxtimes P$, where H has bounded treewidth and P is a path, we obtain $(1 + o(1)) \log n$-bit labelling schemes for all of these graph families. All of these results are optimal up to the lower order term.

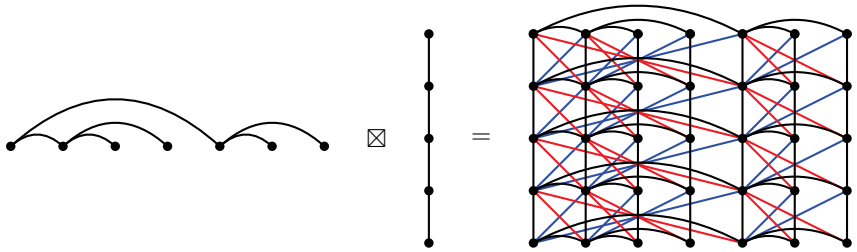

Figure 1. The strong product $H \boxtimes P$ of a tree H and a path P.

A graph is *apex* if it has a vertex whose removal leaves a planar graph. A graph is k-*planar* if it has a drawing in the plane in which each edge is involved in at most k crossings. Such graphs provide a natural generalisation of planar graphs, and have been extensively studied [21]. The definition of k-planar graphs naturally generalises for other surfaces. A graph G is (g, k)-*planar* if it has a drawing in some surface of Euler genus at most g in which each edge of G is involved in at most k crossings. Note that already 1-planar graphs are not minor closed. The generalization of Theorem 1 provided by known product structure theorems is summarized in the following result:

Theorem 2. *For every fixed integer $t \geqslant 1$, the family of all graphs G such that G is a subgraph of $H \boxtimes P$ for some graph H of treewidth t and some path P has a $(1 + o(1)) \log n$-bit adjacency labelling scheme. This includes the following graph classes:*

1) *graphs of bounded genus and, more generally, apex-minor free graphs;*
2) *bounded degree graphs that exclude a fixed graph as a minor; and*
3) *k-planar graphs and, more generally, (g, k)-planar graphs.*

The case of graphs of bounded degree from minor-closed classes (point 2 in Theorem 2) is particularly interesting since, prior to the current work, the best known bound for adjacency labelling schemes in planar graphs of bounded degree was the same as for general planar graphs, i.e., $(4/3 + o(1)) \log n$. On the other hand, our Theorem 2 now gives an asymptotically optimal bound of $(1 + o(1)) \log n$ for graphs of bounded degree from any proper minor-closed class.

C. Outline

The remainder of the paper is organized as follows. Section II reviews some preliminary definitions and easy results. Section III describes a new type of balanced binary search tree that has the specific properties needed for our application. Section IV solves a special case, where G is an n-vertex subgraph of $P_1 \boxtimes P_2$ where P_1 and P_2 are both paths. We include it to highlight the generic idea behind our adjacency labelling scheme. Section V solves the general case in which G is an n-vertex subgraph of $H \boxtimes P$ where H has bounded treewidth and P is a path. Section VI concludes with a discussion of the computational complexity of assigning labels and testing adjacency and presents directions for future work.

Due to space limitations, most proofs are omitted in this extended abstract. The full version of the paper is available on arXiv at `https://arxiv.org/abs/2003.04280`.

II. Preliminaries

All graphs we consider are finite and simple. The vertex and edge sets of a graph G are denoted by $V(G)$ and $E(G)$, respectively. The *size* of a graph G is denoted by $|G| := |V(G)|$.

For any graph G and any vertex $v \in V(G)$, let $N_G(v) := \{w \in V(G) : vw \in E(G)\}$ and $N_G[v] := N_G(v) \cup \{v\}$ denote the open neighbourhood and closed neighbourhood of v in G, respectively.

A. Prefix-Free Codes

For a string $s = s_1, \ldots, s_k$, we use $|s| := k$ to denote the length of s. A string $s_1, \ldots, s_k$ is a *prefix* of a string $t_1, \ldots, t_\ell$ if $k \leqslant \ell$ and $s_1, \ldots, s_k = t_1, \ldots, t_k$. A *prefix-free code* $c\colon X \to \{0,1\}^*$ is a one-to-one function in which $c(x)$ is not a prefix of $c(y)$ for any two distinct $x, y \in X$. Let $\mathbb{N}$ denote the set of non-negative integers. The following is a classic observation[4] of Elias from 1975.

Lemma 3 (Elias [17])**.** *There exists a prefix-free code $\gamma\colon \mathbb{N} \to \{0,1\}^*$ such that, for each $i \in \mathbb{N}$, $|\gamma(i)| \leqslant 2\lfloor \log(i+1) \rfloor + 1$.*

In the remainder of the paper, γ (which we call an *Elias encoding*) will be used extensively, without referring systematically to Lemma 3.

B. Labelling Schemes Based on Binary Trees

A *binary tree* T is a rooted tree in which each node except the root is either the *left* or *right* child of its parent and each

[4]Observing that if the binary writing of an integer $i \geqslant 1$ is w, where w is a word of $\lfloor \log i \rfloor$ bits, then the code $0^{|w|}, 1, w$, i.e., the length of w in unary followed by 1 and then i in binary, is prefix-free.

node has at most one left and at most one right child. For any node x in T, $P_T(x)$ denotes the path from the root of T to x. The *length* of a path P is the number of edges in P, i.e., $|P| - 1$. The *depth*, $d_T(x)$ of x is the length of $P_T(x)$. The *height* of T is $h(T) := \max_{x \in V(T)} d_T(x)$. A *perfectly balanced* binary tree is any binary tree T of height $h(T) = \lfloor \log |T| \rfloor$.

A binary tree is *full* if each non-leaf node has exactly two children. For a binary tree T, we let T^+ denote the full binary tree obtained by attaching to each node x of T $2 - c_x$ leaves where $c_x \in \{0, 1, 2\}$ is the number of children of x. We call the leaves of T^+ the *external nodes* of T. (Note that none of these external nodes are in T.)

A node a in T is a *T-ancestor* of a node x in T if $a \in V(P_T(x))$. If a is a T-ancestor of x then x is a *T-descendant* of a. (Note that a node is a T-ancestor and T-descendant of itself.) For a subset of nodes $X \subseteq V(T)$, the *lowest common T-ancestor* of X is the maximum-depth node $a \in V(T)$ such that a is a T-ancestor of x for each $x \in X$.

Let $P_T(x_r) = x_0, \ldots, x_r$ be a path from the root x_0 of T to some node x_r (possibly $r = 0$). Then the *signature* of x_r in T, denoted $\sigma_T(x_r)$ is a binary string $b_1, \ldots, b_r$ where $b_i = 0$ if and only if x_i is the left child of x_{i-1}. Note that the signature of the root of T is the empty string.

A *binary search tree* T is a binary tree whose node set $V(T)$ consists of distinct real numbers and that has the *binary search tree property*: For each node x in T, $z < x$ for each node z in x's left subtree and $z > x$ for each node z in x's right subtree. For any $x \in \mathbb{R} \setminus V(T)$, the *search path* $P_T(x)$ in T is the unique root-to-leaf path $v_0, \ldots, v_r$ in T^+ such that adding x as a (left or right, as appropriate) child of v_{r-1} in T would result in a binary search tree T' with $V(T') = V(T) \cup \{x\}$.

The following observation allows us to compare values in a binary search tree just given their signatures in the tree.

Observation 4. *For any binary search tree T and any nodes x, y in T, we have $x < y$ if and only if $\sigma_T(x)$ is lexicographically less than $\sigma_T(y)$.*

Let $\mathbb{R}^+$ denote the set of positive real numbers. The following is a folklore result about biased binary search trees.

Lemma 5. *For any finite $S \subset \mathbb{R}$ and any function $w\colon S \to \mathbb{R}^+$, there exists a binary search tree T with $V(T) = S$ such that, for each $y \in S$, $d_T(y) \leqslant \log(W/w(y))$, where $W := \sum_{y \in S} w(y)$.*

The following fact about binary search trees is useful, for example, in the deletion algorithms for several types of balanced binary search trees [22, Section 6.2.3], see Figure 2:

Observation 6. *Let T be a binary search tree and let x, y be nodes in T such that $x < y$ and there is no node z in T such that $x < z < y$, i.e., x and y are consecutive in the sorted order of $V(T)$. Then*

1) *(if y has no left child) $\sigma_T(x)$ is obtained from $\sigma_T(y)$ by removing all trailing 0's and the last 1; or*
2) *(if y has a left child) $\sigma_T(x)$ is obtained from $\sigma_T(y)$ by appending a 0 followed by $d_T(y) - d_T(x) - 1$ 1's.*

Therefore, there exists a function $D\colon (\{0,1\}^*)^2 \to \{0,1\}^*$ such that, for every binary search tree T and for every two consecutive nodes x, y in the sorted order of $V(T)$, there exists $\delta_T(y) \in \{0,1\}^*$ with $|\delta_T(y)| = \mathcal{O}(\log h(T))$ such that $D(\sigma_T(y), \delta_T(y)) = \sigma_T(x)$.

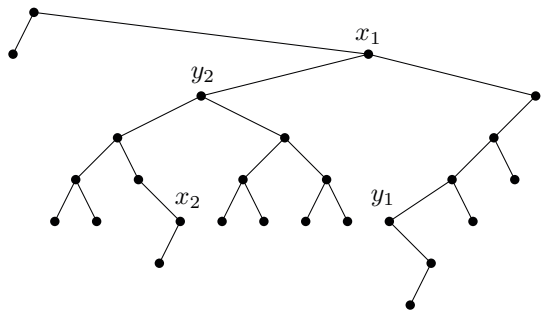

Figure 2. An illustration of Observation 6: (1) $\sigma_T(y_1) = 11000$ and $\sigma_T(x_1) = 1\cancel{1000}$ (2) $\sigma_T(y_2) = 10\cancel{011}$ and $\sigma_T(x_2) = 10011$.

The bitstring $\delta_T(y)$ from Observation 6 is obtained as follows: It consists of a first bit indicating whether y has a left child in T or not and, in case y does have a left child, an Elias encoding $\gamma(s)$ of the value $s := d_T(y) - d_T(x) - 1$. More precisely, $\delta_T(y) = 0$ or $\delta_T(y) = 1, \gamma(s)$.

Putting some of the preceding results together we obtain the following useful coding result.

Lemma 7. *There exists a function $A\colon (\{0,1\}^*)^2 \to \{-1, 0, 1, \bot\}$ such that, for any $h \in \mathbb{N}$, and any $w\colon \{1, \ldots, h\} \to \mathbb{R}^+$ there is a prefix-free code $\alpha\colon \{1, \ldots, h\} \to \{0,1\}^*$ such that*

1) *for each $i \in \{1, \ldots, h\}$, $|\alpha(i)| = \log W - \log w(i) + \mathcal{O}(\log \log h)$, where $W := \sum_{j=1}^{h} w(j)$;*
2) *for any $i, j \in \{1, \ldots, h\}$,*

$$A(\alpha(i), \alpha(j)) = \begin{cases} 0 & \text{if } j = i; \\ 1 & \text{if } j = i + 1; \\ -1 & \text{if } j = i - 1; \\ \bot & \text{otherwise.} \end{cases}$$

C. Chunked Sets

For non-empty finite sets $X, Y \subset \mathbb{R}$ and an integer a, we say that X *a-chunks* Y if, for any $a+1$-element subset $S \subseteq Y$, there exists $x \in X$, such that $\min S \leqslant x \leqslant \max S$. Observe that, if X a-chunks Y, then $|Y \setminus X| \leqslant a(|X|+1) \leqslant 2a|X|$ so $|X \cup Y| \leqslant (2a+1)|X|$. A sequence $V_1, \ldots, V_h \subset \mathbb{R}$

is *a-chunking* if V_y a-chunks V_{y+1} and V_{y+1} a-chunks V_y for each $y \in \{0, \dots, h-1\}$.

Lemma 8. *For any finite sets $S_1, \dots, S_h \subset \mathbb{R}$, there exist sets $V_1, \dots, V_h \subset \mathbb{R}$ such that*

1) for each $y \in \{1, \dots, h\}$, $V_y \supseteq S_y$;
2) $V_1, \dots, V_h$ is 3-chunking;
3) $\sum_{y=1}^{h} |V_y| \leqslant 2 \sum_{y=1}^{h} |S_y|$.

A proof of a much more general version of Lemma 8 (with larger constants) is implicit in the iterated search structure of Chazelle and Guibas [11]. For the sake of completeness, the full version of the paper includes a proof of Lemma 8 that borrows heavily from the amortized analysis of partially persistent data structures [13, Section 2.3].

D. Product Structure Theorems

The *strong product* $A \boxtimes B$ of two graphs A and B is the graph whose vertex set is the Cartesian product $V(A \boxtimes B) := V(A) \times V(B)$ and in which two distinct vertices (x_1, y_1) and (x_2, y_2) are adjacent if and only if:

1) $x_1 x_2 \in E(A)$ and $y_1 y_2 \in E(B)$; or
2) $x_1 = x_2$ and $y_1 y_2 \in E(B)$; or
3) $x_1 x_2 \in E(A)$ and $y_1 = y_2$.

Theorem 9 (Dujmović et al. [15])**.** *Every planar graph G is a subgraph of a strong product $H \boxtimes P$ where H is a graph of treewidth at most 8 and P is a path.*

Theorem 9 can be generalized (replacing 8 with a larger constant) to bounded genus graphs, and more generally to apex-minor free graphs.

Dujmović, Morin, and Wood [16] gave analogous product structure theorems for some non-minor closed families of graphs including k-planar graphs, powers of bounded-degree planar graphs, and k-nearest-neighbour graphs of points in $\mathbb{R}^2$. Dujmović, Esperet, Morin, Walczak, and Wood [14] proved that a similar product structure theorem holds for graphs of bounded degree from any (fixed) proper minor-closed class. This is summarized in the following theorem:

Theorem 10 ([15],[14],[16])**.** *Every graph G in each of the following families of graphs is a subgraph of a strong product $H \boxtimes P$ where P is a path and H is a graph of bounded treewidth:*

- *graphs of bounded genus and, more generally, apex-minor free graphs;*
- *bounded degree graphs that exclude a fixed graph as a minor;*
- *k-planar graphs and, more generally, (g, k)-planar graphs.*

III. Bulk Trees

Our labelling scheme for a subgraph G of $H \boxtimes P$ uses labels that depend in part on the rows (H-coordinates) of G, where each row corresponds to one vertex of P: Say P consists of vertices $1, 2, \dots, h$ in this order, then the *i-th row* of G is the subgraph H_i of G induced by the vertex set $\{(v, i) \in V(G)\}$. A naive approach to create labels for each H_i is to use a labelling scheme for bounded treewidth graphs; roughly, this entails building a specific binary search tree T_i and mapping each vertex v of H_i onto a node x of T_i that we call the *position* of v in T_i. The label of (v, i) encodes the position of v in T_i plus some small extra information (see Section V). This way, we can determine if two vertices (v, i) and (w, i) in the same row are adjacent.

The key problems that we face here though are queries of the type (v, i) and $(w, i+1)$: We would like to determine adjacency on the H-coordinate using T_i or T_{i+1}. We could extend the node set of T_{i+1} so that it represents all vertices from H_i. This way we know that both v and w are represented in T_{i+1}. However, we still have a major issue: the label of (v, i) describes the position of v in T_i but not in T_{i+1}. In this setup, in order to determine if v and w are adjacent in H we need to know their respective positions in the same binary search tree. However, there is in principle no relation between the position of v in T_i and its position in T_{i+1}.

To circumvent this difficulty, we build the binary search trees $T_1, \dots, T_h$ one by one, starting with a balanced binary search tree, in such a way that T_{i+1} is obtained from T_i by performing carefully structured changes. By storing some small extra information related to these changes in the label of (v, i), this will allow us to obtain the position of v in T_{i+1}. Finally, we also need to guarantee that the binary search trees in our sequence are balanced so that the labels are of length $\log |T_i|$ plus a lower-order term.

In this section, we introduce three operations on a binary search tree that will allow us to carry out this plan. These operations are called *bulk insertion*, *bulk deletion*, and *rebalancing*. Starting from a perfectly balanced binary search tree T_1, each tree T_i in our sequence $T_1, \dots, T_h$ will be obtained from T_{i-1} by applying these three operations.

A. Bulk Insertion

The bulk insertion operation, BulkInsert(I), in which a finite set $I \subset \mathbb{R} \setminus V(T)$ of new values are inserted into a binary search tree T, is implemented as follows: Let $z_0, \dots, z_{|T|}$ denote the external nodes of T. For each $i \in \{0, \dots, |T|\}$, let I_i consist of all $x \in I$ such that $P_T(x)$ ends at z_i. For each $i \in \{0, \dots, |T|\}$, construct a perfectly balanced binary search tree T_i with vertex set I_i. For each $i \in \{1, \dots, |T|\}$, replace z_i with T_i in T^+. The resulting tree is the outcome of the operation.

Lemma 11. *Let T be any binary search tree and let I be a finite set of values from $\mathbb{R} \setminus V(T)$ such that $V(T)$ 3-chunks I.*[5] *Apply* BulkInsert(I) *to T to obtain T'. Then T' is a*

[5]There is nothing special about the constant 3 here. The data structure and its analysis work with 3 replaced by any constant a. The constant 3 comes from an application of Lemma 8.

supergraph of T and $h(T') \leqslant h(T) + 2$.

Lemma 12. *Let T be any binary search tree and let I be a finite set of values from $\mathbb{R} \setminus V(T)$ such that $V(T)$ 3-chunks I. Apply* BULKINSERT(I) *to T to obtain T'. Let x be any node of T and let T_x and T'_x be the subtrees of T and T', respectively, rooted at x. Then $|T_x| \leqslant |T'_x| \leqslant 8|T_x|$.*

B. Bulk Deletion

The bulk deletion operation, BULKDELETE(D), of a subset D of nodes of a binary search tree T is implemented as a series of $|D|$ individual deletions, performed in any order. For each $x \in D$, the deletion of x is implemented by running the following recursive algorithm: If x is a leaf, then simply remove x from T. Otherwise, x has at least one child. If x has a left child, then recursively delete the largest value x' in the subtree of T rooted at the left child of x and then replace x with x'. Otherwise x has a right child, so recursively delete the smallest value x' in the subtree of T rooted at the right child of x and then replace x with x'.

Lemma 13. *Let T be any binary search tree and let D be a finite set of values from $V(T)$. Apply* BULKDELETE(D) *to T to obtain a new tree T'. Then, for any node x in T', $\sigma_{T'}(x)$ is a prefix of $\sigma_T(x)$. In particular, $h(T') \leqslant h(T)$.*

Lemma 14. *Let T be any binary search tree and let D be a finite set of values from $V(T)$ such that $V(T) \setminus D$ 3-chunks D. Apply* BULKDELETE(D) *to T to obtain a new tree T'. Then $|T|/8 \leqslant |T'| \leqslant |T|$.*

C. Rebalancing

The rebalancing operation on a binary search tree T uses several subroutines that we now discuss, beginning with the most fundamental one: SPLIT(x).

1) SPLIT(x)*:* The argument of SPLIT(x) is a node x in T and the end result of the subroutine is to split T into two binary search trees $T_{<x}$ and $T_{>x}$ where $V(T_{<x}) = \{z \in V(T) : z < x\}$ and $V(T_{>x}) = \{z \in V(T) : z > x\}$. Refer to Figure 3. Let $P_T(x_r) = x_0, \ldots, x_r$ be the path in T from the root x_0 of T to $x = x_r$. Partition $x_0, \ldots, x_{r-1}$ into two subsequences $a := a_1, \ldots, a_s$ and $b := b_1, \ldots, b_t$ where the elements of a are less than x and the elements of b are greater than x. Note that the properties of a binary search tree guarantee that

$$a_1 < \cdots < a_s < x < b_t < \cdots < b_1.$$

Make a binary search tree T_0 that has x as root, the path $a_1, \ldots, a_s$ as the left subtree of x and the path $b_1, \ldots, b_t$ as the right subtree of x. Note that a_{i+1} is the right child of a_i for each $i \in \{1, \ldots, s-1\}$ and b_{i+1} is the left child of b_i for each $i \in \{1, \ldots, t-1\}$.

Next, consider the forest $F := T - \{x_0, \ldots, x_r\}$. This forest consists of $r + 2$ (possibly empty) trees $A_1, \ldots, A_{r-1}, L, R$ where L and R are the subtrees of T rooted at the left and right child of x in T_x and, for each $i \in \{1, \ldots, r-1\}$, A_i is the subtree of T rooted at the child $c_i \neq x_{i+1}$ of x_i (if such a child exists, otherwise A_i is empty). Make a binary search tree T_x by replacing each of the $r + 2$ external nodes of T_0^+ with the corresponding tree in F. Finally, let $T_{<x}$ be the subtree of T_x rooted at the left child of x and let $T_{>x}$ be the subtree of T_x rooted at the right child of x in T_x.

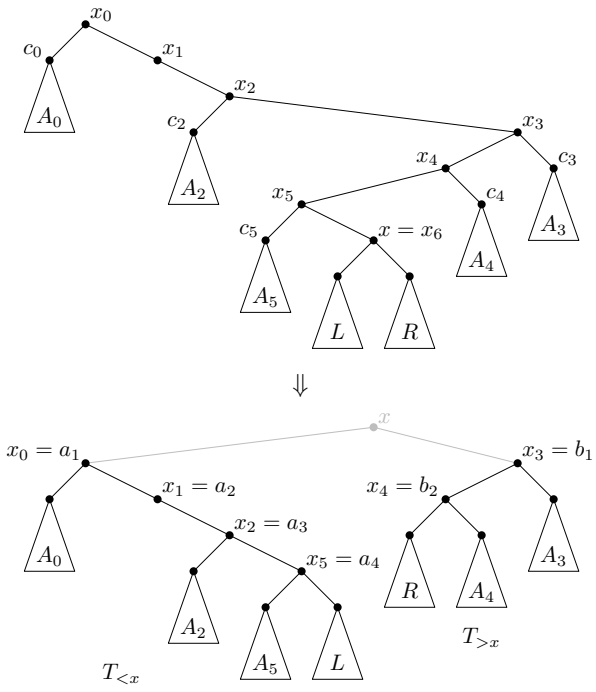

Figure 3. The operation of SPLIT(x).

Lemma 15. *Let T be any binary search tree, let x be any node of T, and apply* SPLIT(x) *to obtain $T_{<x}$ and $T_{>x}$. Then $h(T_{<x}) \leqslant h(T)$ and $h(T_{>x}) \leqslant h(T)$.*

The following observation shows that there is a simple relationship between a node's signature in T before calling SPLIT(x) and its signature in $T_{<x}$ or $T_{>x}$.

Observation 16. *Let T, x, $x_0, \ldots, x_r$, $A_1, \ldots, A_{r-1}, L, R$, $a_1, \ldots, a_s$, and $b_1, \ldots, b_t$ be defined as above. Then*

1) for each $j \in \{1, \ldots, s\}$ where $a_j = x_i$
   a) $\sigma_{T_{<x}}(a_j) = 1^{j-1}$, and
   b) $\sigma_{T_{<x}}(z) = 1^{j-1}, 0, \sigma_{A_i}(z)$ for each $z \in V(A_i)$;

2) for each $j \in \{1, \ldots, t\}$ where $b_j = x_i$
   a) $\sigma_{T_{>x}}(b_j) = 0^{j-1}$, and
   b) $\sigma_{T_{>x}}(z) = 0^{j-1}, 1, \sigma_{A_i}(z)$ for each $z \in V(A_i)$;

3) $\sigma_{T_{<x}}(z) = 1^s, \sigma_L(z)$ for each $z \in V(L)$; and

4) $\sigma_{T_{>x}}(z) = 0^t, \sigma_R(z)$ for each $z \in V(R)$.

In particular, for any $z \in V(T) \setminus \{x\}$, $\sigma_{T_{<x}}(z)$ or $\sigma_{T_{>x}}(z)$

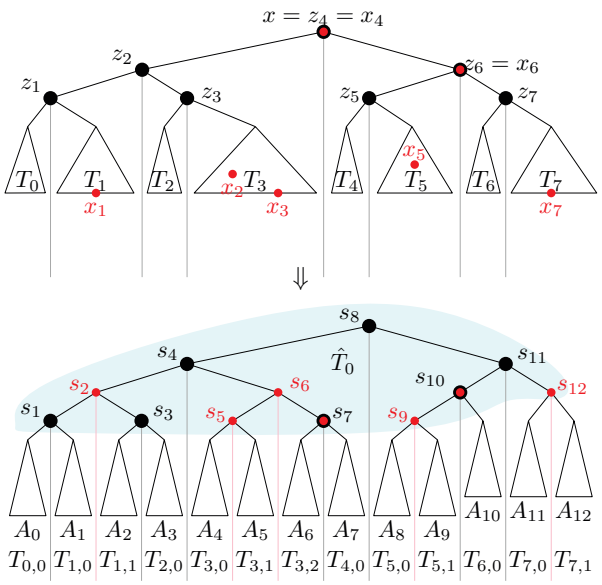

Figure 4. The operation of BALANCE(x, k)

can be obtained from $\sigma_T(z)$ by deleting a prefix and replacing it with one of the $4 \cdot h(T)$ strings in $\Pi := \bigcup_{j=0}^{h(T)-1}\{0^j, 0^j1, 1^j, 1^j0\}$.

2) MULTISPLIT$(x_1, \ldots, x_c)$*:* From the SPLIT(x) operation we build the MULTISPLIT$(x_1, \ldots, x_c)$ operation that takes as input a sequence of nodes $x_1 < \cdots < x_c$ of T. For convenience, define $x_0 = -\infty$ and $x_{c+1} = \infty$. The effect of MULTISPLIT$(x_1, \ldots, x_c)$ is to split T into a sequence of binary search trees $T_0, \ldots, T_c$ where, for each $i \in \{0, \ldots, c\}$, $V(T_i) = \{z \in V(T) : x_i < z < x_{i+1}\}$.

The implementation of MULTISPLIT$(x_1, \ldots, x_c)$ is straightforward divide-and-conquer: If $c = 0$, then there is nothing to do. Otherwise, call SPLIT$(x_{\lceil c/2 \rceil})$ to obtain $T_{<x_{\lceil c/2 \rceil}}$ and $T_{>x_{\lceil c/2 \rceil}}$. Next, apply MULTISPLIT$(x_1, \ldots, x_{\lceil c/2 \rceil - 1})$ to $T_{<x_{\lceil c/2 \rceil}}$ to obtain $T_0, \ldots, T_{\lceil c/2 \rceil - 1}$ and then apply MULTISPLIT$(x_{\lceil c/2 \rceil + 1}, \ldots, x_c)$ to $T_{>x_{\lceil c/2 \rceil}}$ to obtain $T_{\lceil c/2 \rceil}, \ldots, T_c$.

The following lemma is immediate from Lemma 15.

Lemma 17. *Let T be any binary search tree and apply* MULTISPLIT$(x_1, \ldots, x_c)$ *to T to obtain $T_0, \ldots, T_c$. Then $h(T_i) \leqslant h(T)$ for each $i \in \{0, \ldots, c\}$.*

3) BALANCE(x, k)*:* The BALANCE(x, k) operation operates on the subtree T_x of T rooted at some node x in T. The goal of this operation is to balance the size of all the subtrees rooted at nodes of depth $d_T(x) + k + 1$ and contained in T_x. Refer to Figure 4.

If $|V(T_x)| < 2^k$, then this operation simply replaces T_x with a perfectly balanced binary search tree containing $V(T_x)$. Otherwise, let $Z := \{z \in V(T_x) : d_{T_x}(z) < k\}$. Call the $m \leqslant 2^k - 1$ elements of Z $z_1 < z_2 < \cdots < z_m$ and, for convenience, define $z_0 = -\infty$ and $z_{m+1} = \infty$.

Select the nodes $X := \{x_1, \ldots, x_{2^k-1}\}$ of T_x where each x_j has rank $\lfloor j|V(T_x)|/2^k \rfloor$ in $V(T_x)$.[6] The BALANCE(x, k) operation will turn T_x into a tree with a top part $\hat{T}_0$ that is a perfectly balanced binary search tree on $Z \cup X$. We now describe how this is done.

$T_x - Z$ is a forest consisting of $m + 1 \leqslant 2^k$ trees $T_0, \ldots, T_m$. (Some of these trees may be empty.) Order $T_0, \ldots, T_m$ so that, for each $i \in \{0, \ldots, m\}$ and each $x' \in V(T_i)$, $z_i < x' < z_{i+1}$. For each $i \in \{0, \ldots, m\}$, let $\{x_{i,1}, \ldots, x_{i,c_i}\} := X \cap V(T_i)$ where $x_{i,1} < \cdots < x_{i,c_i}$ and define $x_{i,0} := z_i$ and $x_{i,c_i+1} := z_{i+1}$. Note that for each $i \in \{0, \ldots, m\}$, $c_i \leqslant |X| \leqslant 2^k - 1$.

For each $i \in \{0, \ldots, m\}$, apply MULTISPLIT$(x_{i,1}, \ldots, x_{i,c_i})$ to the tree T_i. As a result of these calls, we obtain sequences of trees $T_{i,0}, \ldots, T_{i,c_i}$ where, for each $i \in \{0, \ldots, m\}$, each $j \in \{0, \ldots, c_i\}$, and each $x' \in V(T_{i,j})$, we have $x_{i,j} < x' < x_{i,j+1}$. Note that if $c_i = 0$ (i.e., if X does not intersect $V(T_i)$), then the result of this call is a single tree $T_{i,0} = T_i$. Observe that $\bigcup_{i=0}^{m}\bigcup_{j=0}^{c_i} V(T_{i,j}) = V(T_x) \setminus (Z \cup X)$.

Let $p := |Z \cup X|$, let $s_1 < \cdots < s_p$ denote the elements of $Z \cup X$ and define $s_0 := -\infty$ and $s_{p+1} := \infty$. For each $\ell \in \{0, \ldots, p\}$, let $i_\ell := |Z \cap \{s_1, \ldots, s_\ell\}|$ and $j_\ell := \ell - \max\{q \in \{1, \ldots, \ell\} : s_q \in Z\}$ and let $A_\ell := T_{i_\ell, j_\ell}$. Then, for each $\ell \in \{0, \ldots, p\}$ and each $x' \in V(A_\ell)$, we have $s_\ell < x' < s_{\ell+1}$.

Now construct a perfectly balanced tree $\hat{T}_0$ with vertex set $V(\hat{T}_0) := \{s_1, \ldots, s_p\} = Z \cup X$. The tree $\hat{T}_0$ has $p + 1$ external nodes $a_0, \ldots, a_p$. We obtain a new tree T'_x by replacing a_ℓ with A_ℓ for each $\ell \in \{0, \ldots, p\}$ in $\hat{T}_0^+$. In the encompassing bulk tree T we replace the subtree T_x with T'_x.

Lemma 18. *Let T be any binary search tree, let x be any node of T, and apply* BALANCE(x, k) *to T to obtain a new tree T'. Then $h(T') \leqslant h(T) + 1$.*

The following statement captures what we win after an application of BALANCE(x, k) to a binary search tree.

Lemma 19. *Let T be any binary search tree, let x be any node of T, let T_x be the subtree of T rooted at x, and apply* BALANCE(x, k) *to T to obtain a new tree T'. Then, for each T'-descendant z of x with $d_{T'}(z) = d_T(x) + k + 1$, the subtree of T' rooted at z has size at most $|T_x|/2^k$.*

4) BULKBALANCE(θ, k)*:* The ultimate restructuring operation in bulk trees is BULKBALANCE(θ, k). It calls BALANCE(x, k) for each node x of depth θ in T. (Note that this operation has no effect if there is no such node.) The following two lemmas are immediate consequences of Lemma 18 and Lemma 19, respectively.

[6]For a finite set $X \subset \mathbb{R}$, and $x \in \mathbb{R}$, the *rank* of x in S is $|\{x' \in S : x' < x\}|$.

Lemma 20. *Let T be any binary search tree and apply the* BULKBALANCE(θ, k) *operation to obtain a new tree T'. Then $h(T') \leqslant h(T) + 1$.*

Lemma 21. *Let T be any binary search tree and apply the* BULKBALANCE(θ, k) *operation to obtain a new tree T'. Let x be any node of T of depth θ and let T_x be the subtree of T rooted at x. Then, for each T'-descendant z of x with $d_{T'}(z) = \theta + k + 1$, the subtree of T' rooted at z has size at most $|T_x|/2^k$.*

D. Bulk Tree Sequences

Let $k \geqslant 7$ be an integer[7] and let $S_0, \dots, S_q$ be a 3-chunking sequence. We define a *one-phase k-bulk tree sequence* based on $S_0, \dots, S_q$ to be a sequence $T_0, \dots, T_q$ of binary search trees such that T_0 is an arbitrary binary search tree on node set S_0 and, for each $y \in \{0, \dots, q-1\}$, we have $h(T_y) > y \cdot (k+1)$ and the tree T_{y+1} is obtained from T_y by applying

(i) BULKBALANCE$(y \cdot (k+1), k)$, then
(ii) BULKINSERT(I) with $I := S_{y+1} \setminus S_y$, and finally
(iii) BULKDELETE(D) with $D := S_y \setminus S_{y+1}$.

Note that $V(T_y) = S_y$ for each $y \in \{0, \dots, q\}$. The sequence is *complete* if $h(T_q) \leqslant q \cdot (k+1)$.

For $k \geqslant 7$ and a 3-chunking sequence $S_1, \dots, S_h$, we define a *k-bulk tree sequence* based on $S_1, \dots, S_h$ to be a sequence $T_1, \dots, T_h$ of binary search trees satisfying: T_1 is a perfectly balanced binary search tree with $V(T_1) = S_1$, and there exist indices $h_1, h_2, \dots, h_\ell$ with $1 = h_1 < h_2 < \cdots < h_\ell = h$ such that $T_{h_j}, T_{h_j+1}, \dots, T_{h_{j+1}}$ is a complete one-phase k-bulk tree sequence based on $S_{h_j}, S_{h_j+1}, \dots, S_{h_{j+1}}$ for each $j \in \{1, \dots, \ell-2\}$, and $T_{h_{\ell-1}}, T_{h_{\ell-1}+1}, \dots, T_{h_\ell}$ is a (non-necessarily complete) one-phase k-bulk tree sequence based on $S_{h_{\ell-1}}, S_{h_{\ell-1}+1}, \dots, S_{h_\ell}$.

Note that if we fix the 3-chunking sequence $S_1, \dots, S_h$, the integer $k \geqslant 7$, and the starting perfectly balanced binary search tree T_1 with $V(T_1) = S_1$, a k-bulk tree sequence based on $S_1, \dots, S_h$ and starting with T_1 exists and is unique. It is obtained by a sequence of one-phase k-bulk tree sequences, where we start a new one-phase sequence as soon as the current one is complete.

This will not be needed until the final sections, but it is helpful to keep in mind that we will ultimately take $k = \max\left\{7, \left\lceil \sqrt{\log n / \log\log n} \right\rceil\right\}$ when considering a k-bulk tree sequence built for our n-vertex graph G, so that the expression $\mathcal{O}(k + k^{-1}\log n)$ (which appears many times in what follows), is $\omega(1)$ and $o(\log n)$.

Lemma 22. *Let $T_0, \dots, T_q$ be a one-phase k-bulk tree sequence. Then, for each $y \in \{0, \dots, q\}$*

(i) $h(T_y) \leqslant h(T_0) + 3y$;
(ii) each subtree of T_y rooted at a node of depth $y \cdot (k+1)$ has size at most $|T_0| \cdot 2^{-y(k-3)}$.

Corollary 23. *Let $T_0, \dots, T_q$ be a one-phase k-bulk tree sequence. Then,*

$$q \leqslant \left\lceil \frac{\log |T_0|}{k-3} \right\rceil.$$

Lemma 24. *Let $T_0, \dots, T_q$ be a complete one-phase k-bulk tree sequence, and let $r_0 := h(T_0) - \log|T_0|$. Then, for each $y \in \{0, \dots, q\}$,*

(i) $|T_0|/8^y \leqslant |T_y|$, and thus $\log|T_0| \leqslant \log|T_y| + 3y$;
(ii) $q = \mathcal{O}(k^{-1}\log|T_y|)$;
(iii) $h(T_y) = \log|T_y| + r_0 + \mathcal{O}(k^{-1}\log|T_y|)$; and
(iv) $h(T_q) = \log|T_q| + \mathcal{O}(k + k^{-1}\log|T_q|)$.

The following lemma shows that trees in a bulk tree sequence are balanced at all times:

Lemma 25. *Let $T_1, \dots, T_h$ be a k-bulk tree sequence and let $y \in \{1, \dots, h\}$. Then*

$$h(T_y) \leqslant \log|T_y| + \mathcal{O}(k + k^{-1}\log|T_y|).$$

E. Transition Codes for Nodes

We now arrive at the *raison d'être* of bulk tree sequences: For two consecutive trees T_y and T_{y+1} in a bulk tree sequence and any $z \in V(T_y) \cap V(T_{y+1})$, the signatures $\sigma_{T_y}(z)$ and $\sigma_{T_{y+1}}(z)$ are so closely related that $\sigma_{T_{y+1}}(z)$ can be derived from $\sigma_{T_y}(z)$ and a short *transition code* $\nu_y(z)$. The following two lemmas make this precise.

Lemma 26. *There exists a function $B : (\{0,1\}^*)^2 \to \{0,1\}^*$ such that, for every binary search tree T, for any integers θ and k with $1 \leqslant \theta \leqslant h(T)$ and $k \geqslant 1$, the following holds. Let T' be the binary search tree obtained by an application of* BULKBALANCE(θ, k) *to T. For each $z \in V(T)$, there exists $\nu(z) \in \{0,1\}^*$ with $|\nu(z)| = \mathcal{O}(k \log h(T))$ such that $B(\sigma_T(z), \nu(z)) = \sigma_{T'}(z)$.*

Lemma 27. *There exists a function $B' : (\{0,1\}^*)^2 \to \{0,1\}^*$ such that, for each k-bulk tree sequence $T_1, \dots, T_h$, each $y \in \{1, \dots, h-1\}$, and each $z \in V(T_y) \cap V(T_{y+1})$, there exists $\nu_y(z) \in \{0,1\}^*$ with $|\nu_y(z)| = \mathcal{O}(k \log h(T_y))$ such that $B'(\sigma_{T_y}(z), \nu_y(z)) = \sigma_{T_{y+1}}(z)$.*

IV. SUBGRAPHS OF $P \boxtimes P$

Before continuing, we show that using the techniques developed thus far, we can already solve a non-trivial special case. In particular, we consider the case in which G is an n-vertex subgraph of $P_1 \boxtimes P_2$ where P_1 is a path on m vertices and P_2 is a path on h vertices. Thus, we identify each vertex of G with a point $(x, y) \in \{1, \dots, m\} \times \{1, \dots, h\}$ in the $m \times h$ grid with diagonals, and G is just a subgraph of this grid, see Figure 5. Obviously, we may assume that $m \leqslant n$ and $h \leqslant n$.

Our motivation for considering this special case is expository: The vertices of P_1 are integers $1, \dots, m$ that can

[7] $k \geqslant 7$ is a technical requirement, to make sure that some inequalities hold later on.

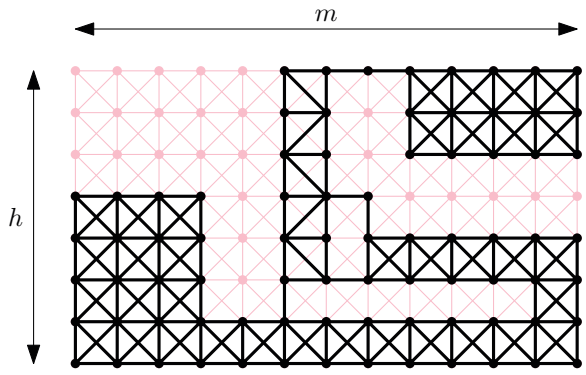

Figure 5. The special case where G is a subgraph of $P_1 \boxtimes P_2$.

be stored directly in a binary search tree. This makes it easier to understand the role that bulk tree sequences play in our solution. The extension of this solution to subgraphs of $H \boxtimes P$, which is the topic of Section V, uses exactly the same ideas but requires another level of indirection since there is no natural mapping from the vertices of H onto real numbers.

A. The Labels

For each $y \in \{1, \ldots, h\}$, we let

$$L_y = \{x : (x, y) \in V(G)\}, \text{ and}$$
$$L_y^+ = L_y \cup \{x - 1 : (x, y) \in V(G)\}.$$

Note that $\sum_{y=1}^{h} |L_y| = n$ and $\sum_{y=1}^{h} |L_y^+| \leqslant 2n$. Let $L_0^+ := \emptyset$.

Let $V_1, \ldots, V_h$ be the 3-chunking sequence obtained by applying Lemma 8 to the sequence $L_1^+ \cup L_0^+, \ldots, L_h^+ \cup L_{h-1}^+$. Thus for each $y \in \{1, \ldots, h\}$, we have

$$V_y \supseteq L_y^+ \cup L_{y-1}^+, \text{ and}$$
$$\textstyle\sum_{y=1}^{h} |V_y| \leqslant 2 \sum_{y=1}^{h} |L_y^+ \cup L_{y-1}^+| \leqslant 8n.$$

Next, let $T_1, \ldots, T_h$ be a k-bulk tree sequence based on $V_1, \ldots, V_h$ (recall that if we fix the starting perfectly balanced binary search tree T_1 with vertex set V_1, this sequence exists and is unique). We discuss the asymptotically optimal choice for the value of k at the end of the section. By Lemma 25, for each $y \in \{1, \ldots, h\}$, we have

$$h(T_y) = \log |T_y| + \mathcal{O}(k + k^{-1} \log |T_y|)$$
$$\leqslant \log |T_y| + \mathcal{O}(k + k^{-1} \log n).$$

Let $A : (\{0,1\}^*)^2 \to \{0,1\}^*$ be the function, given by Lemma 7 such that using the weight function $w(y) := |T_y|$ for each $y \in \{1, \ldots, h\}$, we have a prefix-free code $\alpha : \{1, \ldots, h\} \to \{0,1\}^*$ such that

$$|\alpha(y)| = \log \left(\textstyle\sum_{i=1}^{h} |T_i| \right) - \log |T_y| + \mathcal{O}(\log \log h)$$
$$\leqslant \log n - \log |T_y| + \mathcal{O}(\log \log n),$$

for each $y \in \{1, \ldots, h\}$, and $A(\alpha(i), \alpha(j))$ outputs 0, 1, -1, or $\perp$, depending whether the value of j is i, $i+1$, $i-1$, or some other value, respectively.

Let $B' : (\{0,1\}^*)^2 \to \{0,1\}^*$ be the function, given by Lemma 27, such that for each $y \in \{1, \ldots, h-1\}$ and each $x \in L_y \subseteq V(T_y) \cap V(T_{y+1})$, there exists a code $\nu_y(x)$ with $|\nu_y(x)| = \mathcal{O}(k \log h(T_y)) = \mathcal{O}(k \log \log n + k \log k)$ such that $B'(\sigma_{T_y}(x), \nu_y(x)) = \sigma_{T_{y+1}}(x)$.

Let $D : (\{0,1\}^*)^2 \to \{0,1\}^*$ be the function, given by Observation 6, such that for every binary search tree T, and every i such that $i - 1$ and i are in T, there exists $\delta_T(i) \in \{0,1\}^*$ with $|\delta_T(i)| = \mathcal{O}(\log h(T))$ such that $D(\sigma_T(i), \delta_T(i)) = \sigma_T(i-1)$.

Finally, given a vertex $v = (x, y)$ of G, we define an array $a(v)$ of 8 bits indicating whether each of the edges between (x, y) and $(x \pm 1, y \pm 1)$ are present in G. Note that some of these 8 vertices may not even be present in G in which case the resulting bit is set to 0 since the edge is not present in G.

Now, in the labelling scheme for G, each vertex $v = (x, y) \in V(G)$ receives a label that is the concatenation of the following bitstrings:

(P1) $\alpha(y)$;
(P2) $\gamma(|\sigma_{T_y}(x)|)$, $\sigma_{T_y}(x)$;
(P3) $\delta_{T_y}(x)$;
(P4) if $y \neq h$ then $1, \delta_{T_{y+1}}(x)$;
if $y = h$ then 0;
(P5) if $y \neq h$ then $1, \nu_y(x)$;
if $y = h$ then 0; and
(P6) $a(v)$.

Two major components of this label are $\alpha(y)$, of length $\log n - \log |T_y| + \mathcal{O}(\log \log n)$, and $\sigma_{T_y}(x)$, of length $\log |T_y| + \mathcal{O}(k + k^{-1} \log n)$. Together they have length $\log n + \mathcal{O}(k + k^{-1} \log n + \log \log n)$. The lengths of the remaining components are as follows: $\gamma(|\sigma_{T_y}(x)|)$, $\delta_{T_y}(x)$, and $\delta_{T_{y+1}}(x)$ have length $\mathcal{O}(\log \log n + \log k)$, $\nu_y(x)$ has length $\mathcal{O}(k \log \log n + k \log k)$, and $a(v)$ has length $\mathcal{O}(1)$. Thus, in total the label has length $\log n + \mathcal{O}(k \log \log n + k \log k + k^{-1} \log n)$.

B. Adjacency Testing

First note that from a given label of $v = (x, y) \in V(G)$, we can decode each block of the label. This is because $\alpha(y)$ is prefix-free, $\gamma(|\sigma_{T_y}(x)|)$ is prefix-free so when we read it we know how long is $\sigma_{T_y}(x)$ and we can isolate it as well. The δ-codes are prefix-free again and $\nu_y(x)$ can be decoded as outlined in the proof of Lemma 27. Finally, the last 8-bits correspond to $a(v)$.

Given the labels of two vertices $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$ in G we can test if they are adjacent as follows.

Looking up the value of $A(\alpha(y_1), \alpha(y_2))$, we determine which of the following applies:

1) $|y_1 - y_2| \geqslant 2$: In this case we immediately conclude that v_1 and v_2 are not adjacent in G since they are not

adjacent even in $P_1 \boxtimes P_2$.

2) $y_1 = y_2$: In this case, let $y := y_1 = y_2$. If the two bitstrings $\sigma_{T_y}(x_1)$, $\sigma_{T_y}(x_2)$ are the same, we conclude that $x_1 = x_2$ and $y_1 = y_2$, so $v_1 = v_2$ and we should output that they are not adjacent. Otherwise, we lexicographically compare $\sigma_{T_y}(x_1)$ and $\sigma_{T_y}(x_2)$. Without loss of generality, $\sigma_{T_y}(x_1)$ is smaller than $\sigma_{T_y}(x_2)$. Therefore, by Observation 4, $x_1 < x_2$. Recall that $x_2 \in L_y$ and $L_y^+ \subseteq V(T_y)$, so $x_2 - 1 \in V(T_y)$. We compute $D(\sigma_{T_y}(x_2), \delta_{T_y}(x_2)) = \sigma_{T_y}(x_2 - 1)$. If $\sigma_{T_y}(x_2 - 1) \neq \sigma_{T_y}(x_1)$, then we immediately conclude that $x_2 < x_1 - 1$, so v_1 and v_2 are not adjacent in G, since they are not adjacent even in $P_1 \boxtimes P_2$. Otherwise, we know that $v_1 = (x_2 - 1, y)$ and $v_2 = (x_2, y)$ are adjacent in $P_1 \boxtimes P_2$. Now we use the relevant bit of $a(v_1)$ (or $a(v_2)$) to determine if v_1 and v_2 are adjacent in G.

3) $y_1 = y_2 - 1$: In this case, we compute $B'(\sigma_{T_{y_1}}(x_1), \nu_{y_1}(x_1)) = \sigma_{T_{y_2}}(x_1)$. Let $y := y_2$. If the two bitstrings $\sigma_{T_y}(x_1)$, $\sigma_{T_y}(x_2)$ are the same, we conclude that $x_1 = x_2$. Thus $v_1 = (x_1, y - 1)$ and $v_2 = (x_1, y)$ are adjacent in $P_1 \boxtimes P_2$. Now we look up the relevant bit of $a(v_1)$ (or $a(v_2)$) to determine if v_1 and v_2 are adjacent in G. Otherwise, we lexicographically compare $\sigma_{T_y}(x_1)$ and $\sigma_{T_y}(x_2)$. If $\sigma_{T_y}(x_1)$ is smaller than $\sigma_{T_y}(x_2)$, then we conclude that $x_1 < x_2$. Recall that $x_2 \in L_y$ and $L_y^+ \subseteq V(T_y)$, so $x_2 - 1 \in V(T_y)$. We compute $D(\sigma_{T_y}(x_2), \delta_{T_y}(x_2)) = \sigma_{T_y}(x_2 - 1)$. If $\sigma_{T_y}(x_2 - 1) \neq \sigma_{T_y}(x_1)$, then we immediately conclude that v_1 and v_2 are not adjacent in G, since they are not adjacent even in $P_1 \boxtimes P_2$. Otherwise, we know that $v_1 = (x_2 - 1, y - 1)$ and $v_2 = (x_2, y)$ are adjacent in $P_1 \boxtimes P_2$. Now we use the relevant bit of $a(v_1)$ (or $a(v_2)$) to determine if v_1 and v_2 are adjacent in G. If $\sigma_{T_y}(x_1)$ is larger than $\sigma_{T_y}(x_2)$, then we conclude that $x_1 > x_2$. Recall that $x_1 \in L_{y-1}$ and $L_{y-1}^+ \subseteq V(T_y)$, so $x_1 - 1 \in V(T_y)$. We compute $D(\sigma_{T_y}(x_1), \delta_{T_y}(x_1)) = \sigma_{T_y}(x_1 - 1)$. If $\sigma_{T_y}(x_1 - 1) \neq \sigma_{T_y}(x_2)$, then we immediately conclude that v_1 and v_2 are not adjacent in G, since they are not adjacent even in $P_1 \boxtimes P_2$. Otherwise, we know that $v_1 = (x_1, y - 1)$ and $v_2 = (x_1 - 1, y)$ are adjacent in $P_1 \boxtimes P_2$. Now we use the relevant bit of $a(v_1)$ (or $a(v_2)$) to determine if v_1 and v_2 are adjacent in G.

4) $y_2 = y_1 - 1$: In this case, we compute $B'(\sigma_{T_{y_2}}(x_2), \nu_{y_2}(x_2)) = \sigma_{T_{y_1}}(x_2)$. Now we proceed as in the previous case.

This establishes our first result:

Theorem 28. *The family $\mathcal{G}$ of n-vertex subgraphs of a strong product $P \boxtimes P$ where P is a path has a $(1 + o(1)) \log n$-bit adjacency labelling scheme.*

Remark 29. The $o(\log n)$ term in the label length of Theorem 28 is $\mathcal{O}(k \log\log n + k \log k + k^{-1} \log n)$. An asymptotically optimal choice of k is therefore $k = \max\left\{7, \left\lceil \sqrt{\log n / \log\log n} \right\rceil\right\}$, yielding labels of length $\log n + \mathcal{O}\left(\sqrt{\log n \log\log n}\right)$.

V. Subgraphs of $H \boxtimes P$

In this section we briefly explain how to extend the ideas of the previous section to subgraphs of $H \boxtimes P$ where H is a graph of treewidth t and P is a path.

It is well known that any m-vertex graph H of treewidth at most t is a subgraph an interval graph of clique number $O(t \log m)$ (i.e. such that any point lies in $O(t \log m)$ intervals). What we will do is consider such an interval representation of H and store all the endpoints of the intervals in a bulk tree T. We then map each vertex v of H onto the deepest node $x_T(v)$ in T that contains both of v's endpoints in its subtree. Consider an edge vw in H, then the intervals of v and w intersect and it follows that there is one root-to-leaf path in T that contains both $x_T(v)$ and $x_T(w)$, so $\sigma(x_T(v))$ is a prefix of $\sigma(x_T(w))$, or vice-versa. This is what makes adjacency testing possible.

Recall that we are considering an n-vertex subgraph G of $H \boxtimes P$. This implies as before that the subgraphs of G induced by two consecutive rows of $H \boxtimes P$ might be two different subgraphs of H, and thus the bulk trees storing the endpoints of the intervals in some interval representation of these two subgraphs of H could be completely different. As before we can use Lemma 8 to allow for a smooth transition between consecutive bulk trees, but a new issue appears. For a particular vertex v in H, the bulk tree operations may cause the value of $x_T(v)$ to change. By carefully inspecting the effect of all three bulk tree operations (in particular rebalancing, which turns out to be quite technical), we can show that this change can be encoded in $o(\log n)$ bits, as before.

The adjacency testing is now fairly similar as before, except that when we have identified that two vertices are in the same row or in two consecutive rows, we have to test for adjacencies in H. Previously we only had to look at the coordinates (did they differ by at most 1?) and a constant number of bits (saying whether the corresponding edge was really there in G compared to $P \boxtimes P$). We now replace this by a variant of an adjacency labelling scheme of Gavoille and Labourel [18] for graphs of bounded treewidth that is tailored to our use of interval representations and bulk trees.

We obtain the following result.

Theorem 30. *For every fixed $t \in \mathbb{N}$, the family of all graphs G such that G is a subgraph of $H \boxtimes P$ for some t-tree H and some path P has a $(1 + o(1)) \log n$-bit adjacency labelling scheme.*

Theorem 1 and Theorem 2 are immediate consequences of Theorem 30, Theorem 9, and Theorem 10.

VI. Conclusion

We conclude with a few remarks on the computational complexity of our labelling scheme. Given an n-vertex planar graph G, finding an 8-tree H , a path P, and a mapping of G into a subgraph of $H \boxtimes P$ can be done in $\mathcal{O}(n \log n)$ time [23]. The process of computing the labels of $V(G)$ as described in Section IV and Section V has a straightforward $\mathcal{O}(n \log n)$ time implementation. Thus, the adjacency labels described in Theorem 1 are computable in $\mathcal{O}(n \log n)$ time for n-vertex planar graphs.

In the discussion on the adjacency test function below, we focus on the case where t is a constant (which is the case in all our applications). The adjacency testing function can then be implemented in time $\mathcal{O}(k)$ in the standard w-bit word RAM model, providing for binary words of length $w = \Omega(\log n)$, bitwise logical operations, bitwise shift operations, and a most-significant-bit operation[8]. We note that Theorem 30 holds whenever k ranges from $\omega(1)$ to $\mathcal{O}(\sqrt{\log n / \log \log n})$. This yields a trade-off between adjacency test time and label length complexities. On one side, by choosing $k = \omega(1)$, we have labels of $(1+o(1)) \log n$ bits and an adjacency test running in nearly constant time. On the other side, by selecting an adjacency test time complexity of $k = \mathcal{O}(\sqrt{\log n / \log \log n})$, the label length is minimized and has length $\log n + \mathcal{O}(\sqrt{\log n \log \log n})$.

The current result leaves two obvious directions for future work:

1) The precise length of the labels in Theorem 1 and Theorem 30 is, at best, $\log n + \mathcal{O}(\sqrt{\log n \log \log n})$. The only known lower bound is $\log n + \Omega(1)$. Closing the gap in the lower-order term remains an open problem.
2) Theorem 30 implies a $(1 + o(1)) \log n$-bit labelling schemes for any family of graphs that excludes an apex graph as a minor. Can this be extended to any K_t-minor free family of graphs?

Acknowledgement

Part of this research was conducted during the Eighth Workshop on Geometry and Graphs, held at the Bellairs Research Institute, January 31–February 7, 2020. We are grateful to the organizers and participants for providing a stimulating research environment. The authors are particularly grateful to Tamara Mchedlidze and David Wood for helpful discussions.

References

[1] Mikkel Abrahamsen, Stephen Alstrup, Jacob Holm, Mathias Bæk Tejs Knudsen, and Morten Stöckel. Near-optimal induced universal graphs for bounded degree graphs. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 128:1–128:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[2] David Adjiashvili and Noy Rotbart. Labeling schemes for bounded degree graphs. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 375–386. Springer, 2014.

[3] Noga Alon. Asymptotically optimal induced universal graphs. *Geometric and Functional Analysis*, 27(1):1–32, February 2017.

[4] Stephen Alstrup, Søren Dahlgaard, and Mathias Bæk Tejs Knudsen. Optimal induced universal graphs and adjacency labeling for trees. *J. ACM*, 64(4):27:1–27:22, 2017.

[5] Stephen Alstrup, Søren Dahlgaard, Mathias Bæk Tejs Knudsen, and Ely Porat. Sublinear distance labeling. In Piotr Sankowski and Christos D. Zaroliagis, editors, *24th Annual European Symposium on Algorithms, ESA 2016, August 22-24, 2016, Aarhus, Denmark*, volume 57 of *LIPIcs*, pages 5:1–5:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[6] Stephen Alstrup, Cyril Gavoille, Esben Bistrup Halvorsen, and Holger Petersen. Simpler, faster and shorter labels for distances in graphs. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 338–350. SIAM, 2016.

[7] Stephen Alstrup, Inge Li Gørtz, Esben Bistrup Halvorsen, and Ely Porat. Distance labeling schemes for trees. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 132:1–132:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[8] Stephen Alstrup, Haim Kaplan, Mikkel Thorup, and Uri Zwick. Adjacency labeling schemes and induced-universal graphs. *SIAM J. Discrete Math.*, 33(1):116–137, 2019.

[9] Stephen Alstrup and Theis Rauhe. Improved labeling scheme for ancestor queries. In David Eppstein, editor, *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA*, pages 947–953. ACM/SIAM, 2002.

[10] Marthe Bonamy, Cyril Gavoille, and Michał Pilipczuk. Shorter labeling schemes for planar graphs. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 446–462. SIAM, 2020.

[11] Bernard Chazelle and Leonidas J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(2):133–162, 1986.

[8] The only purpose of the most-significant-bit operation is allow decoding of the Elias γ code in constant time.

[12] Fan R. K. Chung. Universal graphs and induced-universal graphs. *Journal of Graph Theory*, 14(4):443–454, 1990.

[13] James R. Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, 1989.

[14] Vida Dujmović, Louis Esperet, Pat Morin, Bartosz Walczak, and David R. Wood. Clustered 3-colouring graphs of bounded degree. *CoRR*, abs/2002.11721, 2020.

[15] Vida Dujmović, Gwenaël Joret, Piotr Micek, Pat Morin, Torsten Ueckerdt, and David R. Wood. Planar graphs have bounded queue-number. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 862–875. IEEE Computer Society, 2019.

[16] Vida Dujmović, Pat Morin, and David R. Wood. The structure of k-planar graphs. *CoRR*, abs/1907.05168, 2019.

[17] Peter Elias. Universal codeword sets and representations of the integers. *IEEE Trans. Information Theory*, 21(2):194–203, 1975.

[18] Cyril Gavoille and Arnaud Labourel. Shorter implicit representation for planar graphs and bounded treewidth graphs. In Lars Arge, Michael Hoffmann, and Emo Welzl, editors, *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, volume 4698 of *Lecture Notes in Computer Science*, pages 582–593. Springer, 2007.

[19] Sampath Kannan, Moni Naor, and Steven Rudich. Implicit representation of graphs. In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 334–343. ACM, 1988.

[20] Sampath Kannan, Moni Naor, and Steven Rudich. Implicit representation of graphs. *SIAM J. Discrete Math.*, 5(4):596–603, 1992.

[21] Stephen G. Kobourov, Giuseppe Liotta, and Fabrizio Montecchiani. An annotated bibliography on 1-planarity. *Comput. Sci. Rev.*, 25:49–67, 2017.

[22] Pat Morin. *Open Data Structures*. Athabasca University Press, 2013.

[23] Pat Morin. A fast algorithm for the product structure of planar graphs. *CoRR*, abs/2004.02530, 2020.

[24] John H. Muller. *Local Structure in Graph Classes*. PhD thesis, School of Information and Computer Science, March 1988.

[25] C. St. J. A. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, 36(1):445–450, 1961.

[26] Jeremy P. Spinrad. *Efficient graph representations*, volume 19 of *Fields Institute monographs*. American Mathematical Society, 2003.