Succinct Planar Encoding with Minor Operations

Frank Kammer
THM, University of Applied Sciences Mittelhessen, Giessen, Germany

Johannes Meintrup
THM, University of Applied Sciences Mittelhessen, Giessen, Germany

Abstract

Let G be an unlabeled planar and simple n-vertex graph. Unlabeled graphs are graphs where the label-information is either not given or lost during the construction of data-structures. We present a succinct encoding of G that provides induced-minor operations, i.e., edge contractions and vertex deletions. Any sequence of such operations is processed in $O(n)$ time in the word-RAM model. At all times the encoding provides constant time (per element output) neighborhood access and degree queries. Optional hash tables extend the encoding with constant expected time adjacency queries and edge-deletion (thus, all minor operations are supported) such that any number of edge deletions are computed in $O(n)$ expected time. Constructing the encoding requires $O(n)$ bits and $O(n)$ time. The encoding requires $\mathcal{H}(n) + o(n)$ bits of space with $\mathcal{H}(n)$ being the entropy of encoding a planar graph with n vertices. Our data structure is based on the recent result of Holm et al. [ESA 2017] who presented a linear time contraction data structure that allows to maintain parallel edges and works for labeled graphs, but uses $\Theta(n \log n)$ bits of space. We combine the techniques used by Holm et al. with novel ideas and the succinct encoding of Blelloch and Farzan [CPM 2010] for arbitrary separable graphs. Our result partially answers the question raised by Blelloch and Farzan whether their encoding can be modified to allow modifications of the graph.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases planar graph, r-division, separator, succinct encoding, graph minors

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.44

Related Version *Full Version*: `https://arxiv.org/abs/2301.10564` [18]

Funding *Johannes Meintrup*: Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 379157101.

1 Introduction

Graphs are used to model systems as entities and relationships between these entities. Many graphs that arise in real-world application are very large. This has given rise to an area of research with the aim of reducing the required space [1, 7, 8, 12, 14, 19]. Practical examples include large road-networks [26] or social-network graphs [10]. This has spawned research inquiries into compact representation of graphs, especially those that posses certain structural properties. The arguably most well-known such structural property is planarity. A graph is planar if it can be drawn in the plane without crossings. In this work we consider the problem of maintaining a succinct encoding of a given graph under edge contractions and vertex deletions, referred to as *induced-minor operations*. An edge contraction in a graph $G = (V, E)$ consists of removing an edge $\{u, v\} \in E$ from the graph and merging its endpoints to a new vertex x. Edge contractions are a vital technique in a multitude of algorithms, prominent examples include computing minimum cuts [20], practical treewidth computations [27] and maximum matchings [6]. At the end of the paper we extend our result from induced-minor operations to support all *minor operations*, which in addition to edge contractions and vertex deletions includes edge deletions. We work on unlabeled graphs, meaning that labels are either not given or lost when constructing our data structure.

Related Work. For encoding planar graphs without regards to providing fast access operations, Keeler et al. [21] showed an $O(n)$ bits representation. For a compression within the information theoretic lower bound refer to He et al. [13]. Due to Munro and Raman [24] there exists an encoding using $O(n)$ bits that allows constant-time queries which has subsequently been improved by Chiang et al. [5]. We build on the succinct representation due to Blelloch and Farzan, which allows encoding arbitrary separable graphs and subsequently allows constant-time queries [3], which builds on the work of Blanford et al. [2]. Recently it was shown that the encoding of Blelloch and Farzan can be constructed using $O(n)$ bits in $O(n)$ time [17]. Our work can be thought of as extending the encoding of Blelloch and Farzan to allow contraction operations. For edge contractions in planar graphs without regard to space-usage, Klein and Mozes [22] presented an algorithm that runs in $O(\log n)$ time per contraction. Their work is based on techniques by Brodal and Fagerberg [4]. For edge and vertex deletions Holm and Rotenberg showed a data structure that provides any number of such deletions in $O(n)$ time [16]. The state-of-the-art by Holm et al. [15] provides edge contractions in $O(n)$ time total. Their data structure allows constant time (per element output) neighborhood and degree queries and maintains parallel edges that occur due to merges, while also allowing to view the graph as "simple", i.e., skipping parallel edges when querying the neighborhoods. Using optional hashing techniques they provide expected constant time adjacency queries. Their data structure is based on the well-known notion of r-divisions [9, 11, 23], a technique we use as well. The data structure of Holm et al. uses $\Theta(n \log n)$ bits to store mappings and initially applies graph transformations that increase the number of vertices by a constant factor, making it not succinct, neither for unlabeled nor labeled graphs. Even assuming these steps could trivially be adapted to use $o(n)$ bits when encoding unlabeled graphs, they additionally construct a lookup table for storing small graphs such that each index encoding a graph with k vertices uses $\mathcal{H}(k) + \Theta(\mathcal{H}(k))$ bits of space, with $\mathcal{H}(k)$ the entropy of encoding planar graphs with k vertices. They then encode graphs of at least n total vertices using such indices, i.e., they use $\mathcal{H}(n) + \Theta(\mathcal{H}(n))$ bits for storing all such small graph, i.e., their data structure is not succinct.

Our result. Let G be an unlabeled planar and simple graph with n vertices. We present a succinct encoding of G that is able to process edge contractions and vertex deletions in $O(n)$ time for any number of such modifications. At all times, the data structure allows constant time (per element output) neighborhood and degree queries. Using optional hashing techniques, we provide constant expected time adjacency queries and can process any number of edge deletions in $O(n)$ expected time. This partially answers a question posed by Blelloch and Farzan if their encoding for arbitrary separable graphs can be extended to allow graph modifications [3]. Our data structure maintains the running time of the state-of-the-art solution by Holm et al. [15] for labeled planar graphs, while using significantly less space. The data structure of Holm et al. requires the input graph to be transformed by replacing each vertex with degree larger than a constant by a cycle-gadget, which increases the number of vertices by a linear factor[1]. Such a transformation is not necessary using our techniques, but our data structure only works for unlabeled graphs, and we are not able to maintain parallel edges that occur due to contractions, i.e., our graph is at all times simple. In the next section we present our main result in an intuitive fashion. Our new result is based on several previous data structures (Section 3) and on a table-lookup technique (Section 4). In Section 5, we describe and extend a succinct encoding technique due to Blelloch and

[1] Outlined in Appendix A.1 in the full version of their paper found on arXiv.

Farzan [3]. One of our challenges was to extend mappings used by Blelloch and Farzan to be semi-dynamic. Interestingly, for this we repurpose a graph data structure of Holm et al. [15] and use it to construct dynamic mappings between vertex labels. The dynamic mappings are described in Section 6, which we use in Section 7 to combine the results of all the previous sections to achieve our dynamic encoding. We end this publication by extending our encoding to provide vertex and edge deletions as well as adjacency and degree queries. All proofs can be found in the full version of this paper.

2 A succinct graph encoding for edge contractions

We now give an overview of our new data structure for succinctly encoding a simple unlabeled planar graph $G = (V, E)$ with n vertices and maintaining this encoding under edge contractions while allowing neighborhood and degree queries in constant time per element output. Vertex deletions and edge deletions are discussed in Section 7. Note that while we work with unlabeled graphs, internally we assume vertices are labeled as consecutive integers from 1 to n. The intuitive idea is to construct a set $X \subset V$ of *boundary vertices* such that $G[V \setminus X]$ (the vertex-induced graph on $V \setminus X$) contains multiple connected components $C_1, \ldots, C_k$ of "small" size at most r (which is defined later) with $k = O(n/r)$, and X of size $O(n/\sqrt{r})$. Based on this, we distinguish edges of three types: edges between vertices of X (called *boundary edges*) edges between vertices of one C_i (called *simple edges*) and edges between a vertex of X and a vertex of some C_i (called *mixed edges*). For each C_i denote with P_i the graph induced by all simple edges between vertices of C_i and all mixed edges with one endpoint in C_i. The set of all these P_i is known as an *r-division*, and each P_i is called a *piece*. Note that an r-division has some additional characteristics which are defined more precisely later, one such key characteristics that each P_i contains only $O(\sqrt{r})$ boundary vertices, and therefore is of size $O(|C_i| + \sqrt{r}) = O(r)$. Note that each boundary vertex is contained in multiple pieces. Assume that there is a data structure that is able to easily contract any number of edges in graphs of size $O(r)$ in time $O(r)$. As long as contractions are only carried out between simple edges, we would be able to easily construct a data structure that results in $O(n)$ runtime for any number of contractions by simply constructing this data structure for each piece individually. Problems occur when contracting boundary or mixed edges because contractions are no longer able to be carried out locally in a single piece as they affect vertices in multiple pieces. For example, when contracting a boundary edge $\{u, v\}$ this affects every P_i with $i \in \{1, \ldots, k\}$ that contains at least one of u and v. To provide such contractions we construct a data structure sketched in the following. Note that we distinguish between vertex merges of two vertices u, v and edge contractions between an edge $\{u, v\}$, with the latter being analogous to the first with the distinction that u must be adjacent to v.

For edges between vertices of the boundary X we construct a *boundary graph* $F = G[X]$, which at all times contains all boundary edges, including edges that occur due to contractions. A key invariant that we uphold is that the "status" of a vertex, i.e., if it is a boundary or a non-boundary vertex, never changes due to contractions. If a vertex is a boundary vertex initially, it will be handled internally as a boundary vertex even when it no longer is incident to boundary edges due to contractions. For non-boundary vertices we ensure that these vertices are never incident to boundary edges. For each $u \in X$ we maintain a mapping containing all i with $u \in V(P_i)$, i.e., the pieces that contains u. Now, when we contract a boundary edge $\{u, v\}$ we firstly merge v to u in all P_i that contain both u and v. For any $u \in V$ we denote with $N(u)$ the neighborhood of u. We process the aforementioned merge by

setting $N(u) := (N(u) \cup N(v)) \setminus \{u, v\}$ in P_i and removing v from P_i. In all P_i that contain only v, we simply relabel v to be u. In all P_i that do not contain v (but can contain u) we do not have to make modifications. Finally, $\{u, v\}$ is contracted in the boundary graph F as well. To maintain F we can use a "slow" data structure, as F is small.

To contract a mixed edge $\{u, v\}$ with $u \in X$ we know there is only one P_i that contains v, where we execute the contraction. This might result in u now being adjacent to some $x \in X \cap V(P_i)$, for which we add the respective edge $\{u, x\}$ to F. It helps to achieve our runtime goal of $O(n)$ for processing any number of edge contractions if we do not add this edge $\{u, x\}$ to all other pieces that contain both u and x. We therefore maintain a second invariant: that edges between boundary vertices are only contained in the boundary graph (for now ignoring the specifics of how this is maintained).

To handle contractions inside pieces, intuitively we build the same data structure we outlined here to one more time, splitting each piece in tiny pieces of size at most r', specified later. We can categorize all graphs of size at most r' by a lookup table, which allows us to encode every such tiny piece as an index into the lookup table (Section 4). For each graph of the lookup table we pre-compute all possible vertex merges. We then contract edges in constant time by simply retrieving (the index of) the contracted graph from the lookup table. Such a framework was previously used by Holm et al. [15] to maintain a planar graph under contractions, but with a space usage of $\Theta(n \log n)$ bits.

To output the neighborhood of a vertex we distinguish between two cases. First, to output the neighborhood of a vertex u that is not a boundary vertex, we can simply output the neighborhood in the only P_i that contains u. For a boundary vertex u we first output the neighborhood of u in F, which are all neighbors being boundary vertices, and then do the same for each piece P_i that contains neighbors of u.

To achieve a succinct data structure we build on a succinct encoding due to Blelloch and Farzan [3], outlined in Section 5. They construct an r-division for the input graph, and for each piece P_i of the r-division construct another r'-division. Each piece $P_{i,j}$ of this r'-division is categorized by a succinct index in a lookup table. They use succinct dictionaries to translate information between the two levels. These translation data structures and the lookup table can be realized with $o(n)$ bits. We use the encoding of Blelloch and Farzan, but enhance it with dynamic qualities (Section 5 and Section 6). This (partially) answers the question posed by Blelloch and Farzan [3], if modifications of the succinctly encoded graph are possible. A key notion is that most of the novel "building blocks" handle the small number of boundary vertices, so non-space efficient data structures are used.

▶ **Theorem 1.** *Let $\mathcal{H}(n)$ be the entropy of encoding a planar graph with n vertices and G an unlabeled simple n-vertex planar graph. There exists an encoding of G that provides induced-minor operations (i.e., vertex deletions and edge contractions) with these properties: The encoding requires $O(n)$ time to execute any number of induced-minor operations and provides neighborhood and degree operations in constant time (per element output). The encoding requires $\mathcal{H}(n) + o(n)$ bits can be initialized in $O(n)$ time and $O(n)$ bits.*

Using optional hashing techniques adapted from the data structure of Holm et al. [15, Lemma 5.15] we provide edge deletion and adjacency queries.

▶ **Corollary 2.** *The encoding of Theorem 1 can be extended to provide expected $O(1)$ time adjacency queries and process any number of minor operations in expected $O(n)$ time.*

3 Preliminaries

We denote by $[k] = \{1, \ldots, k\}$, with k any integer. Our model of computation is the word-RAM with a word length of $w = \Omega(\log n)$ bits. We work with simple unlabeled graphs. In the following let $G = (V, E)$. We use $G[V']$ with $V' \subseteq V$ to denote the vertex induced subgraph on V' of G. We also write $V(G)$ for the vertices V of G and $E(G)$ for E. A merge of a pair of two vertices $u, v \in V$ means replacing both vertices u, v with a vertex x with $N(x) = N(u) \cup N(v) \setminus \{u, v\}$. In our data structures we merge u and v by setting $N(u) := N(u) \cup N(v) \setminus \{u, v\}$ and removing v from V, i.e., x is either u or v. We then say that v *is merged to* u. Note that merging u to v is a different operation. Merging two adjacent vertices u, v is called *contracting the edge* $\{u, v\}$. We denote by n the number of vertices of the graph G under consideration. In this work we use n_G to denote the number of vertices of a given graph G. If the graph is clear from the context we simply write n.

Planar graph. A graph is planar exactly if it can be drawn in the plane such that no two edges cross. The family of planar graphs is closed under taking minors. Any simple planar graph G has $O(n)$ edges. We only work with simple graphs and from this point onward assume all (planar) graphs are simple. An operation that modifies a planar graph under consideration is *planarity preserving* if afterwards the graph is still planar. For brevity's sake, we assume all merges discussed in our work are planarity preserving unless stated otherwise. We denote with $\mathcal{H}(\cdot)$ the entropy to encode a planar graph, a function dependent on the number of vertices of the graph under consideration. It is known that $\mathcal{H}(n) = \Theta(n)$ [28].

We assume w.l.o.g. that all input graphs for our data structure are connected. If this is not the case, we add a dummy vertex and connect it to an arbitrary vertex in each connected component. After our encoding of Theorem 1 is constructed, we can simply ignore the dummy vertex and all its incident edges during any output of the provided operations. As we require this modification to be space-efficient, concretely that it only uses $O(n)$ bits of additional space during the construction and runs in $O(n)$ time, we provide an explicit lemma for this modification. Note that a succinct encoding of a planar graph with this additional dummy vertex requires only a constant number of additional bits, as $\mathcal{H}(n+1) = \mathcal{H}(n) + O(1)$.

▶ **Lemma 3.** *Let G be a simple planar graph. We can add a dummy vertex v_d to V and all edges $\{v_d, u\}$ to E with u being one vertex in each connected component of G using $O(n)$ bits in $O(n)$ time.*

Graph divisions. Let $G = (V, E)$ be a planar graph and r some integer. An *r-division* $\mathcal{R} = \{P_1, \ldots, P_k\}$ of G is a division of G into $k = O(n/r)$ edge-disjoint connected subgraphs called *pieces*. Each piece has $O(r)$ vertices. For each $P \in \mathcal{R}$ there exists a set of *boundary vertices* $\delta P \subset V(P)$ such that $u \in \delta P$ if and only if u is incident to an edge $\{u, v\} \in E$ with $v \notin V(P)$. For each P it holds that $|\delta P| = O(\sqrt{r})$. We denote with $\delta\mathcal{R} = \bigcup_{P \in \mathcal{R}} \delta P$ the set of boundary vertices of $\mathcal{R}$. For any r-division $\mathcal{R}$ we denote by k the number of pieces, and assume they are numbered from $[k]$ as $\mathcal{R} = P_1, \ldots, P_k$. We use a subscript numbering to distinguish between multiple r-divisions as follows: we use the same subscript numbering to refer to the number of parts of the r-division, i.e., we use k_i when talking about an r-division $\mathcal{R}_i$ for some integer i. Linear time algorithms for computing r-divisions exists [11, 23]. Note that an r-division requires each piece to be connected, and our encoding in some sense maintains a dynamic r-division. Due to modifications, pieces may become disconnected. We do not require each piece to be connected once the encoding is constructed, this abuses the definition of r-divisions without consequence.

Forbidden-vertex graph data structure. We use a so-called forbidden-vertex graph data structure that is initialized for a simple planar graph $G = (V, E)$ and any set $B \subseteq V$ of *forbidden vertices*. It allows modifications of G in the form of edge insertions and deletions, and in the form of vertex merges while maintaining two invariants: no edges between vertices of B exist and G is simple and planar. This data structure was described by Holm et al. [15] as a building block for their contraction data structure. We slightly modify their data structure and change some notation to match our use-cases. We use this data structure (among other things) for maintaining edges between so-called boundary vertices, as sketched in Section 2. We refer to this data structure as *forbidden-vertex graph data structure*.

For each vertex and edge managed by the data structure we can access and modify auxiliary data, which takes constant time per word written or read, if a reference to the vertex or edge is given. When merging two vertices u, v some edge $\{u, x\}$ might be removed and inserted again as $\{v, x\}$. We view this as the same edge, but with different endpoints. Meaning, auxiliary data of $\{u, x\}$ is now stored at $\{v, x\}$. If $\{v, x\}$ already existed before the merge, we can decide what to do with the data of the discarded parallel edge. Self-loops and forbidden edges that would occur due to a merge are output during the merge operation. All operations are only permitted if they preserve planarity. In the following we more precisely define the available modifications:

- **Merge.** Given are two vertices u, v with $u \neq v$. Merge v to u by setting $N(u) := N(u) \cup N(v) \setminus \{u, v\}$ and removing v and all incident edges from V. Returns a reference to u and reports and discards all parallel edges during the merge, and reports all non-parallel edges inserted to $N(u)$ during the merge. Edges that would occur between vertices of B are discarded.
- **Insert.** Given are two vertices u, v with $u \neq v$ and $\{u, v\} \notin E$. Insert the edge $e = \{u, v\}$ into E, unless both $u, v \in B$.
- **Delete.** Let $\{u, v\}$ be a given edge. Remove the edge $\{u, v\}$ from E.

▶ **Lemma 4** ([15]). *Let $G = (V, E)$ be a simple planar graph and $B \subseteq V$. A forbidden vertex graph data structure can be initialized for G and B in $O(n \log n)$ time. It provides constant time (per element output) neighborhood and adjacency queries and access to the label mappings. Edge insertion/deletion takes $O(\log n)$ time. Any number of free-assignment vertex merges are executed in $O(n \log^2 n)$ time. The data structure uses $O(n \log n)$ bits.*

To achieve the runtime outlined in Lemma 4 for vertex merges, each merge of two vertices $u, v \in V$ is processed by merging the vertex with the lowest degree to the vertex with the highest degree, i.e., we can not freely choose which vertex is merged. A simple mapping using standard data structures allows us to label the vertex $x \in \{u, v\}$ that remains after the merge either u or v. The details of this are found in full version of our paper. We refer to a merge of two vertices where we are able to freely decide the labeling of the remaining vertex after the merge as *free-assignment merge*. We henceforth assume that all merges of the data structure are free assignment merges. The following lemma summarizes this.

▶ **Lemma 5.** *Let $G = (V, E)$ be a simple planar graph and $B \subseteq V$ managed by the forbidden-vertex graph data structure of Lemma 4. Using $O(n)$ additional time for initialization and $O(n \log n)$ bits we are able to process any number of free-assignment merges in $O(n \log^2 n)$ time on G.*

Indexable dictionaries. We use a data structure called *indexable dictionary* (ID), initialized for a universe U of consecutive integers and a set $S \subseteq U$ and supports membership, rank and select queries. A rank query for some $x \in U$ returns $|\{y \in S : y < x\}|$. A select query for some integer i returns the value of $x \in U$ such that x is stored at rank i.

▶ **Lemma 6** ([25]). *Let $s \leq u$ be two integers. Given a set S of size s, which is a subset of a universe $U = [u]$, there is a succinct indexable dictionary (ID) on S that requires $\log \binom{u}{s} + o(s) + O(\log \log u)$ bits and supports rank/select on elements of S in constant time.*

We use IDs with $u = n$ and $s = O(n/\Lambda(n))$ for some function $\Lambda(n) = \omega(1)$, then each ID requires $o(n)$ bits. IDs can be can be constructed in $O(u)$ time using $O(u)$ bits [25].

4 Table lookup for small planar graphs

In this section we present our table lookup data structure for small graphs. Given an integer ℓ the table lists for every positive integer $\ell' \leq \ell$ every possible planar graph with at most ℓ' vertices. Such a lookup table was used by Blelloch and Farzan [3] as a building block for succinctly encoding planar and other separable graphs, which we outline Section 5. For every graph G encoded by the table, they provide adjacency queries and neighborhood iteration in constant time (per element). The table can be realized using $O(2^{\mathtt{poly}(\ell)})$ bits and time, including the data structures needed to provide the queries. To distinguish between all planar graphs with ℓ' vertices we requires $\mathcal{H}(\ell') + O(1)$ bits. This corresponds to an index into the computed table. The table contains $2^{\mathcal{H}(\ell)}$ entries. Everything mentioned so far was shown by Blelloch and Farzan. In the following we introduce additional operations and extensions of this lookup table. These modifications increase the size of the table by a negligible amount of bits while maintaining the same (asymptotical) runtime for constructing the table. We show that our modifications increase the size of each index encoding a graph with ℓ' vertices by $o(\ell')$ bits, which is negligible for our use case.

- Range filtered neighborhood iteration: Takes as input an index i into the lookup table, three vertices $u, a, b \in V$, with $G = (V, E)$ being the graph encoded at index i and $u \neq a \neq b$. Provides an iterator over all neighbors v of u with $a \leq v \leq b$.
- Batch edge deletion: Takes as input an index i into the lookup table, three vertices $u, a, b \in V$ with $G = (V, E)$ the graph encoded at index i. Returns the index j encoding the graph $G' = (V, E \setminus X)$ with $X = \{\{u, v\} : v \in N(u) \text{ and } a \leq v \leq b\}$.
- Label-preserving merge: Takes as input an index i into the lookup table and two vertices $u, v \in V$, with $G = (V, E)$ being the graph encoded at index i. Returns the index j encoding the graph G' obtained from G by setting $N(u) := N(u) \cup N(v) \setminus \{u, v\}$ and deleting v. Only allowed when preserving planarity.

All these operations can easily be pre-computed for one entry of the table in time $O(\text{poly}(\ell))$ using the same amount of bits. The sum of computations over every entry of the table is $O(2^{\text{poly}(\ell)})$. As a note on the label-preserving merge operation, vis-à-vis keeping the vertex v, but marking it deleted, consider the following example. We want to merge the vertices u and v in some graph $G = (V, E)$ encoded by the table with ℓ' vertices. If we would simply merge them, the vertex v no longer exists. In particular, the vertex set after the merge is $V' = V \setminus \{v\}$. As the vertex set for each graph encoded by the table is consecutively numbered from $[\ell']$, graphs with vertex set V' are possibly not encoded by the table. To remedy this, we simply keep the vertex v in the graph but mark it deleted. It remains to show how to handle 'mark v as deleted'. To each graph encoded by the table we add a dummy vertex called `deleted`. The table lists every possible planar graph with at most $\ell + 1$ vertices, where the vertex with the largest label is our dummy vertex `deleted`. To mark v as deleted during the label preserving merge, we set $N(\texttt{deleted}) := N(\texttt{deleted}) \cup \{v\}$ (and $N(v) := \{\texttt{deleted}\}$). We are now able to check if a vertex is deleted, by checking if it is adjacent to `deleted`. When we later encode a graph $G = (V, E)$ via an index into this lookup table, we encode it as the graph $G' = (V \cup \{\texttt{deleted}\}, E)$, i.e., with no vertices marked as deleted initially.

It remains to observe how one additional extra vertex increases the size of the table. The size of each entry encoded by the table stays asymptotically the same, and is thus of no concern to us. The number of entries in the table is $O(2^{\mathcal{H}(\ell+1)})$, and therefore each index into the lookup table encoding a graph with ℓ' vertices requires $\mathcal{H}(\ell'+1)+O(1)$ bits to be stored. As $\mathcal{H}(\ell'+1)=\Theta(\ell'+1)$ [28] it holds that $\mathcal{H}(\ell'+1)=\mathcal{H}(\ell')+O(1)$, which is negligible for our purpose. Therefore, our table uses $2^{\mathtt{poly}(\ell)}$ bits, which is asymptotically the same as the original table of Blelloch and Farzan. Each index of the table uses only a constant number of additional bits over the theoretical lower bound. Note that indices into the original unmodified table of Blelloch and Farzan also require this additional $O(1)$ bits. We introduce some further modifications that increase the additional space per index storing a graph with r' vertices by $o(r')$ bits, which is fine for our use-case.

When we later use the table lookup to contract edges, we do so by effectively replacing an unlabeled graph with a different unlabeled one. Without care, this can break internal labeling structures, e.g., a vertex in a graph encoded by the lookup table has an internal label of 5, and after replacing the graph it now has a label of 7. Section 6 and Section 7 show how additionally maintain a dynamic label mapping structure for "important" vertices, i.e., boundary vertices as described in Section 2. For this we require the graph encoded by the table to be partially labeled. Concretely this means that the labels remain correct for boundary vertices when replacing one graph with a different one. We store all possible labels for boundary vertices, of which there are $b=O(\sqrt{\ell})$ many. In detail, when encoding a single graph G with ℓ vertices, we do not store a single unlabeled representative of G in the table (i.e., a graph isomorphic to all labeled versions of G), but all graphs such that $\ell-b$ vertices are unlabeled, e.g., have an arbitrary internal label, and b vertices have all possible labelings in the range $0,\dots,\ell$. This increases the size of the table by a negligible factor, outined now. Due to the partial labeling we require, the number of bits needed to store an index into the table increases by $O(\log(\binom{\ell}{b}))=O(\log(\binom{\ell}{\sqrt{\ell}}))=o(\ell)$, and thus is negligible for our use-case. Later, when we modify a graph G_i encoded as an index i of lookup table (e.g., contract edges), we replace the index i with the index j such that the graph G_j encoded by j represents the modified graph with the additional characteristic that all boundary vertices of G_j have the same label in G_i. the labeling for the non-boundary vertices changes due to this, but what we maintain is that a non-boundary vertex remains mapped to non-boundary vertex, and that all boundary vertices maintain their same labeling, which is all that we require for our data-structure. This is expressed via a set of invariants defined in Section 6 and Section 7.

▶ **Lemma 7.** *Let ℓ be a positive integer. There exists a table that encodes all planar graphs with vertex set $\{1,\dots,\ell'\}$ for all integers $\ell'\leq\ell$ with the following properties. For every graph encoded by the table, (range filtered) neighborhood iteration, adjacency queries and label-preserving merge operations and batch edge deletion are provided in constant time (per element). The table can be constructed in $O(2^{\mathrm{poly}(\ell)})$ time using $O(2^{\mathrm{poly}(\ell)})$ bits. Every index of the table referencing a graph with ℓ' vertices requires $\mathcal{H}(\ell')+o(\ell')$ bits.*

5 Succinct encoding of planar graphs

We now describe the succinct encoding of unlabeled planar (and other separable) graphs due to Blelloch and Farzan [3]. We use their data structure as a basis for our encoding. Our result effectively extend their encoding with (induced-) minor operations. For this we need to give a technical overview of their encoding. Let $G=(V,E)$ be an unlabeled planar graph, $\mathcal{R}=\{P_1,\dots,P_k\}$ an r-division with $r=\log^4 n$, and for each P_i with $i\in[k]$,

let $\mathcal{R}_i = \{P_{i,1}, \ldots, P_{i,k}\}$ be an r'-division of P_i with $r' = \log^4 \log^4 n$. [2] The encoding assigns three integer labels to each vertex $u \in V$. A label in the entire graph (called *global label*) a label in each piece $P_i \in \mathcal{R}$ (called a *mini label*) and a label in each piece $P_{i,j} \in \mathcal{R}_i$ (called a *micro label*). We refer to G with the newly assigned labels as the *global graph*, each labeled $P_i \in \mathcal{R}$ as a *mini graph*, and each labeled $P_{i,j} \in \mathcal{R}_i$ as a *micro graph*. Note that boundary vertices of $\delta\mathcal{R}$ receive multiple mini labels, and analogous boundary vertices of $\delta\mathcal{R}_i$ receive multiple micro labels. We refer to the boundary vertices of $\delta\mathcal{R}$ with their assigned global labels as δG, and the set of boundary vertices of $\delta\mathcal{R}_i$ with their assigned mini labels in P_i as δP_i. For a given boundary vertex u identified by its global label, we refer to all occurrences u' (as a mini label) of u in a mini graph P_i as *duplicates*, and the same for boundary vertices of mini graphs P_i in regard to their occurrences in micro graphs. We refer to the set of (mini labels of) duplicate vertices in a mini graph P_i as ΔP_i, and analogous the set of (micro labels of) duplicate vertices in a micro graph $P_{i,j}$ as $\Delta P_{i,j}$. Global labels are consecutive integers assigned first to all non-boundary vertices and then to boundary vertices, i.e., all boundary vertices have larger labels than non-boundary vertices. Analogous for mini labels in mini graphs. Micro labels are assigned arbitrarily. For operations vertices are identified by their respective label. E.g., a neighborhood query of a vertex $u \in V$ takes the global label of u as an input and outputs the global labels of all $v \in N(u)$, analogous for queries in a mini or micro graph. Micro graphs are encoded as an index into a lookup table $\mathcal{T}$, listing all planar graphs of at most r' vertices. Technically this is realized by an array with one entry for each micro graph, which can be indexed by (i, j) when retrieving the entry for micro graph $P_{i,j}$. For our use case we replace the table of Blelloch and Farzan with the table described in Section 4, which provides additional operations. We now describe operations that the encoding provides, which are used by Blelloch and Farzan in their original publication, but are not defined outside of 1. All mappings are implemented using IDs (Lemma 6) over the universe $[n]$ combined with standard data structures such as lists and pointers. Let $u \in V$ be a vertex identified by its global label. For each such u, the encoding provides a mapping to access a list $\phi(u)$ that contains tuples (i, u') with i the index of a mini graph P_i that contains mini label u' of u. The lists are sorted in increasing order by i. Note that if u is a non-boundary vertex the mapping contains only a single tuple. For each such tuple (i, u') we can access a mapping $\phi_i^{-1}(u') = u$. For vertices u' in each mini graph P_i (identified by their mini label) analogous mappings $\phi_i(u')$ containing tuples (j, u'') with j the index of a micro graph $P_{i,j}$ that contains micro label u'' of u', and the analogous mappings $\phi_{i,j}^{-1}(u'') = u'$ are provided. We refer to all these mappings as *static translation mappings*.

Recall that Blelloch and Farzan assign micro labels in an arbitrary fashion. We instead assign the labels according to a coloring we define in the following. Let $P_{i,j}$ be the micro graph we want to label. We first assign labels to vertices that are neither a boundary vertex of $\delta\mathcal{R}$ nor of $\delta\mathcal{R}_i$, which we assign the color `simple`. Then we assign labels to vertices that are in the boundary $\delta\mathcal{R}$, but not in $\delta\mathcal{R}_i$, which we assign the color `global-boundary`. Then to vertices that are not in the boundary $\delta\mathcal{R}$, but in the boundary $\delta\mathcal{R}_i$, colored `mini-boundary`, and finally to vertices in both the boundary $\delta\mathcal{R}$ and in $\delta\mathcal{R}_i$, colored `double-boundary`. Consequently, for any four vertices of $P_{i,j}$ it holds that $a < b < c < d$ if a is colored `simple`, b is colored `global-boundary`, c is colored `mini-boundary` and d is colored `double-boundary`. For each mini graph we store the lowest labeled vertex of each color, which uses negligible space of $O((n/\log^4 \log^4 n) \log \log^4 \log^4 n) = o(n)$ bits overall.

[2] Blelloch and Farzan use $r' = \log n / \log \log n$ in their publication, but make it clear that there is a large degree in freedom as long as the choice is of size $o(\log n)$.

Combined with our novel way of assigning micro labels to vertices of micro graphs, the range-filtered neighborhood operation provided by our table allows us to implement the *color-filtered neighborhood* operation that outputs all neighbors x of a vertex u (in a micro graph $P_{i,j}$) such that all x are colored with $c \in \{\texttt{simple}, \texttt{global-boundary}, \texttt{mini-boundary}, \texttt{double-boundary}\}$. The operation runs in constant time (per element output). Using the label preserving merge operation of the lookup table we can easily provide such merges for every micro graph. Analogous for the vertex and edge deletion operation.

For a given unlabeled planar graph G we refer to the encoding described in this section as *basic encoding of* G. Kammer and Meintrup [17] have shown that the encoding can be constructed in $O(n)$ time using $O(n)$ bits. Our modifications have negligible impact on the runtime and space usage of the construction. This results in the following theorem.

▶ **Theorem 8.** *Let G be an unlabeled planar graph and $\mathcal{H}(n)$ the entropy of encoding a planar graph with n vertices. There exists a basic encoding of G into a global graph, mini graphs and micro graphs that uses $\mathcal{H}(n) + o(n)$ bits total. The basic encoding provides static translation mappings for the global graph, each mini graph and each micro graph. For each micro graph the encoding provides degree, adjacency, (color-filtered) neighborhood, (batch) edge/vertex deletion and label-preserving merge operations in $O(1)$ time. The basic encoding can be constructed in $O(n)$ time using $O(n)$ bits.*

6 Dynamic mapping data structures

For this section assume a planar graph G is given via the basic encoding of Theorem 8. We now describe a set of dynamic mapping structures, for which we outline the use-case in the following. We already mentioned in Section 2 that a vertex that is initially a boundary vertex (globally or/and in mini graphs) will never become a non-boundary vertex, and a non-boundary vertex will never become a boundary vertex due to any of our edge contractions. We construct dynamic variants of the static translation mappings for boundary vertices (in the global or in mini graphs). We ensure that the static translation mappings remain valid for all non-boundary vertices. Later these mappings are concretely constructed for the vertices of the initial graph (i.e., before any contractions are processed) and are maintained for all of these vertices throughout. Concretely this means that the sets for which we define mappings and data structures never change after initialization. Recall from Section 2 that when contracting an edge $\{u, v\} \in E$ we effectively forward the contraction operation to mini graphs that contain mini labels u' and v' of u and v respectively, and then forward the contraction to micro graphs in an analogous way. For our solution we require that these cascading merge operations are handled independently without interfering with each other.

Consider for example the case where we contract an edge $\{u, v\} \in E$ with u being a boundary vertex and v a non-boundary vertex. In this case we want to contract v to u (technical reasons for this are outlined in the next section). To fulfill this contraction, we forward a request to the mini graph P_i to merge the vertices u' and v', the respective mini labels of u and v in P_i. In the case that v' is a boundary vertex in P_i, but u' is a non-boundary vertex in P_i, we want to merge u' to v', which is in conflict with our desire to merge v to u in the global graph. The idea is to support free-assignment merges, whose realization is described in the next paragraph. This sort of conflict only pertains to vertices u that are part of the boundary δG (thus, a duplicate ΔP_i in P_i) and/or have a mini label u' in some P_i that is part of the boundary δP_i, i.e., we need to provide free-assignment merges for vertices of $\delta P_i \cup \Delta P_i$. We construct a dynamic mapping that allows us to decide, when merging two vertices $u', v' \in \delta P_i \cup \Delta P_i$, if the vertex that remains after the merge is

labeled u' or v'. This is realized by assigning each such vertex an *external mini label* and an *internal mini label*. Effectively we have no free choice on which internal mini label the vertex has after a merge, but we are free to assign a new external label. We construct a mapping $\texttt{internal}_i : \delta P_i \cup \Delta P_i \to \delta P_i \cup \Delta P_i$ that maps a given external label of a vertex of $\delta P_i \cup \Delta P_i$ to its internal label, and a mapping $\texttt{external}_i : \delta P_i \cup \Delta P_i \to \delta P_i \cup \Delta P_i$ that maps a given internal label of a vertex $\delta P_i \cup \Delta P_i$ to its external label. For all other vertices of P_i we ensure that the external and internal mini labels are identical, and therefore do not need to construct any mapping. This is explicitly defined in Invariant 2 later in this section.

▶ **Lemma 9.** *All mappings $\texttt{internal}_i$ and $\texttt{external}_i$ can be constructed in $O(n)$ time using $o(n)$ bits of space. They provide read/write access in $O(1)$ time.*

We also have the need for a dynamic version of the static translation mappings ϕ and ϕ_i for boundary vertices δG in the global graph and boundary vertices δP_i in mini graphs, and the mappings ϕ_i^{-1} and $\phi_{i,j}^{-1}$ for the duplicate vertices ΔP_i in mini graphs and $\Delta P_{i,j}$ micro graphs, outlined in the following paragraphs. Initially these are equal to the static mappings. We first describe the dynamic versions of the mappings ϕ_i^{-1} and $\phi_{i,j}^{-1}$, which we refer to as Φ_i^{-1} and $\Phi_{i,j}^{-1}$ respectively. Afterwards we describe the dynamic versions of the mappings ϕ and ϕ_i, referred to as Φ and Φ_i. To give an intuition for the use-case of these mappings, consider a contraction of an edge $e = \{u, v\} \in E$ with $u, v \in \delta G$. We contract this edge by first merging all u' and v' in the mini graphs P_i that contain both the duplicate u' of u and v' of v. In all P_i that contain only a duplicate of v' of v we need to know that the global label of v' is now (i.e., after the contraction) u instead of v, for which we use the described mappings. In all other mini graphs no changes need to be made.

▶ **Lemma 10.** *All mappings $\Phi_i^{-1} : \Delta P_i \to \delta G$ and $\Phi_{i,j}^{-1} : \Delta P_{i,j} \to \delta P_i$ can be constructed in $O(n)$ time using $o(n)$ bits of space. They provide read/write access in $O(1)$ time.*

We now describe the dynamic mappings Φ and Φ_i. As mentioned, we initially require all mappings $\Phi(u)$ to be equal to $\phi(u)$ for $u \in \delta G$ and analogously all mappings $\Phi_i(u')$ to be initially equal to $\phi_i(u')$ for $u' \in \delta P_i$ for all mini graphs P_i. To represent these mappings we construct a graph H that contains all boundary vertices $u \in \delta G$ and, for each mini graph P_i, a vertex p_i, with edges $\{u, p_i\}$ added to H exactly if u has a duplicate u' in P_i. Note that the existance of a duplicate u' in P_i means that u' has a non-boundary neighbor in P_i (initially). At each such edge we store the tuple (i, u'). We construct H using the forbidden vertex graph data structure of Lemma 4. The tuples stored at the incident edges of a vertex $u \in \delta G$ in H are exactly the set $\phi(u)$. Note that H is a minor of G and therefore planar. We can provide for all $u \in \delta G$: iteration over all elements $\Phi(u)$ (by iterating over $N(u)$ in H), insertion and removal of elements in $\Phi_i(u)$ (by inserting or removing edges in H), the merge of two sets $\Phi(u)$ and $\Phi(v)$ for some $v \in \delta G$ (by merging u to v or v to u in H). Some other similar operations are provided, outlined in detail in the next section where we concretely describe our edge contraction algorithm. For each mini graph P_i the analogous graph H_i is constructed, which manages the mappings Φ_i. An important note is that for each tuple $(i, u') \in \Phi(u)$ for $u \in \delta G$ the vertex u' is the external mini label of some vertex in P_i. As no external or internal micro labels are defined for micro graphs, each tuple $(j, u'') \in \Phi_i(u')$ for all mini graphs P_i contains the concrete micro label u'' in $P_{i,j}$. In the next section we describe our neighborhood operation, for which we require a special version of the mappings Φ, which we first motivate with an intuition. To output the neighborhood of a vertex $u \in \delta G$ we (intuitively) iterate over all $(i, u') \in \Phi(u)$ and, for each such (i, u'), iterate (and translate to global labels) over all neighbors of u' in P_i. To achieve a runtime of $O(|N(u)|)$ for this

operation, we require that each tuple (i, u') "contributes" at least one such neighbor. While this is true initially, due to edge contractions (and other modifications) the degree of each such u' can become 0. To remedy this, we store a special version of the mappings $\Phi(u)$ which we refer to as $\Phi^{>0}$, containing only tuples $(i, u') \in \Phi(u)$ such that the degree of u' is > 0. We construct the analogous mappings $\Phi_i^{>0}$ for all P_i. During contractions, we update the mappings $\Phi^{>0}$ and $\Phi_i^{>0}$ to uphold the aforementioned characteristic, which is formalized in Invariant 1. How this invariant is upheld, is discussed in the next section. We realize these mappings exactly as Φ and Φ_i, respectively, i.e., as graphs $H^{>0}$ and $H_i^{>0}$. Initially $H^{>0} = H$ and all $H_i^{>0} = H_i$, by the definition of boundary vertices in r-divisions (Section 3).

▶ **Invariant 1** (non-zero-degree invariant)**.** *For all $u \in \delta G$, each entry $(i, u') \in \Phi^{>0}(u)$ guarantees that u' has degree > 0 in mini graph P_i. For all $u' \in \delta P_i$ over all mini graphs P_i, each entry $(j, u'') \in \Phi_i^{>0}(u')$ guarantees that u'' has degree > 0 in micro graph $P_{i,j}$.*

▶ **Lemma 11.** *Graphs H, $H^{>0}$ and H_i, $H_i^{>0}$ can be constructed in $O(n)$ time and $o(n)$ bits.*

Using all data structures described in this section we uphold invariants below while running contractions on G – the details on this are described in the next section. For better readability we slightly abuse the definition of our `internal` (`external`) mappings of by assuming they return the identity function for $u' \notin \delta P_i \cup \Delta P_i$.

▶ **Invariant 2** (label-translation invariants)**.**

a. Global to external mini label and vice-versa:

I. *For each $u \in V \setminus \delta G$ and $(i, u') = \phi(u)$ it holds that u' is the external mini label of u in P_i and $\phi^{-1}(u') = u$.*

II. *For each $u \in \delta G$ and $(i, u') \in \Phi(u)$ (Φ is the dynamic version of ϕ) it holds that u' is the external mini label of u in P_i and $\Phi^{-1}(u') = u$.*

b. Internal to external mini label and vice-versa:

For each vertex $u' \in V(P_i)$ identified by its external mini label, it holds that $u^* =$ `internal`(u') is the internal mini label of u' and `external`$(u^*) = u'$.

c. Mini to micro label and vice-versa:

I. *For each $u' \in V(P_i) \setminus \delta P_i$ (identified by its internal mini label) and $(j, u'') = \phi_i(u')$ it holds that u'' is the micro label of u' in $P_{i,j}$ and $\phi_{i,j}^{-1} = u'$.*

II. *For each $u^* \in \delta P_i$ (identified by its internal mini label) and for each $(j, u'') \in \Phi_i(u^*)$ it holds that u'' is the micro label of u^* in $P_{i,j}$ and $\Phi_{i,j}^{-1}(u'') = u^*$.*

7 Towards a succinct dynamic encoding

For this section let $G = (V, E)$ be a graph encoded by the basic encoding of Theorem 8. Also, assume that the mappings of Lemma 9, 10 and 11 are constructed and available. In this section we describe our solution to support modifications of G. We denote by $\bar{G} = (\bar{V}, \bar{E})$ the graph G before any modifications are processed, e.g., contractions of edges. Analogously we define by $\bar{P}_i$, $\bar{P}_{i,j}$ the initial mini and micro graphs, respectively, with its initial vertices (as mini/micro labels). As sketched in Section 2 we handle contractions between so-called boundary edges with a *boundary graph* $F = G[\delta G]$ and analogously a *mini boundary graph* $F_i = P_i[\delta P_i]$ for each mini graph P_i. These graphs are realized via the forbidden vertex graph data structure (Lemma 4). For F we use as the set of forbidden vertices the empty set. For each F_i we use the duplicate vertices ΔP_i of P_i as the set of forbidden vertices. During initialization edges between forbidden vertices are removed. The forbidden-vertex graph data structure makes sure that this remains true after initialization. Let $\{u, v\} \in E$

be an edge. We say $\{u, v\}$ is *managed by* F if $\{u, v\} \in E(F)$, $\{u, v\}$ is *managed by* F_i if $\{u', v'\} \in E(F_i)$ with u' and v' the mini labels of u and v respectively, and finally we say $\{u, v\}$ is *managed by* $P_{i,j}$ if it contains the edge $\{u'', v''\}$ with u'', v'' being the micro labels of u and v, respectively. We uphold the following invariant:

▶ **Invariant 3** (edge-singleton invariant)**.**

a. *An edge $\{u, v\}$ is managed by F exactly if $u, v \in \delta G$.*

b. *An edge $\{u, v\}$ is managed by F_i exactly if $u', v' \in \delta P_i \setminus \Delta P_i$, with u', v' the respective mini labels of u and v in P_i. In this case, no other F_j $(j \neq i)$ also manages $\{u, v\}$.*

c. *All edges $\{u, v\}$ not managed by F or some F_i are managed by one micro graph $P_{i,j}$.*

Our construction of F and each F_i is the first step to achieve this invariant. We now give an intuition why this invariant is useful. Due to some edge contractions new edges $\{u, v\}$ might occur in G between boundary vertices (either global boundary vertices or vertices that are boundary vertices in a mini graph). We can not afford to add this edge to all mini and micro graphs that contain mini and micro labels of both u and v, respectively. Instead, we only add this edge to F or some F_i. Moreover, if edges $e \in E$ are managed multiple times, the runtime of the neighborhood operation can increase.

An important note is that the basic encoding of G does not adhere to the edge-singleton invariant from the get-go, i.e., edges managed by some F or F_i might be contained in one or more micro graphs initially. Using the batch edge deletion operation provided for micro graphs (Theorem 8) we can delete all edges that would initially violate our invariant. If this violates Invariant 1, we remove the respective entries from $\Phi^{>0}$ ($\Phi_i^{>0}$). This uses $O(n)$ time.

We refer to the combination of the basic encoding of G, the mappings of Lemma 9, Lemma 10 and Lemma 11, the boundary graph F and each mini boundary graph F_i as *succinct dynamic encoding of* G, summarized in the following corollary.

▶ **Corollary 12.** *The succinct dynamic encoding of G can be constructed in $O(n)$ time using $O(n)$ bits. After construction the encoding requires $\mathcal{H}(n) + o(n)$ bits. The encoding upholds the label-translation, edge-singleton and non-zero degree invariant.*

We now give an intuition how we implement the neighborhood operation for a vertex $u \in V$ identified by its global label. We first output all neighbors of u in F (which is $\emptyset$ if u is not a boundary vertex) and then, for all P_i that contain a mini label u' of u, compute all neighbors v' of u' in F_i (which is again $\emptyset$ if u' is not a boundary vertex) and output the respective global label v of v'. We then go to all micro graphs $P_{i,j}$ that contain a mini label u'' of u', compute all neighbors v'' of u'' in $P_{i,j}$ and output the respective global label v of v''. By the edge singleton invariant it is easy to see that each neighbor v of u in G is output exactly once by this algorithm. Invariant 2 provides the necessary translation operations. The missing details are discussed in the proof of the following lemma.

▶ **Lemma 13.** *For any $u \in V$ the neighborhood operation runs in time $O(|N(u)|)$.*

We now focus on our edge-contraction algorithm. For this we introduce one last invariant, which we call the *status invariant*. As sketched in Section 2 we require that for every vertex being a boundary vertex (either globally or in a mini graph) to remain a boundary vertex, and for every non-boundary vertex to remain a non-boundary vertex. For this we slightly abuse the definition of boundary vertices. By definition of r-divisions (Section 3) a boundary vertex $u \in \delta G$ has neighbors in more than one mini graph. Due to contractions (or other modifications) this might at some point no longer be true. Nonetheless, we still consider such a vertex to be a boundary vertex. We require that a boundary vertex remains a boundary vertex, and a non-boundary vertex remains a non-boundary vertex. For this we maintain the following invariant that depends on our slight abuse of the boundary vertex definition.

▶ **Invariant 4** (status invariant). *For every $u \in V$ and every $u' \in V(P_i)$ over all mini graphs P_i, it holds $u \in \delta G$ if and only if $u \in \delta \bar{G}$ as well as $u' \in \delta P_i$ if and only if $u' \in \delta \bar{P}_i$.*

We now give an overview of our edge-contraction algorithm, which we describe in three levels: vertex merges in micro graphs, vertex merges in mini graphs and edge contractions in the global graph. We guarantee the four invariants (Invariants 1, 2, 3 and 4) before and after each edge contraction. Technically, merges are executed in (mini) boundary graph(s) and micro graphs. Everything else is to maintain the mappings of Section 6. An edge $\{u, v\} \in E$ is contracted by determining all micro graphs $P_{i,j}$ that contain (micro labels of) u and v, all mini boundary graphs F that contain (mini labels of) u and v and check if F contains u and v. In all structures that contain u and v we merge v to u and update the mappings of Section 6. To guarantee the invariants we split the responsibilities among the three levels:

- **Global Graph-Responsibility.** Contractions in the global graph maintain Invariant 1 regarding $\Phi^{>0}$, Invariant 2.a., Invariant 3.a and Invariant 4 for all $u \in V$.
- **Mini Graph-Responsibility.** Vertex merges in a mini graph P_i maintain Invariant 1 regarding $\Phi_i^{>0}$, Invariant 2.b-c, Invariant 3.b, and Invariant 4 for all $u' \in V(P_i)$.
- **Micro Graph-Responsibility.** Vertex merges in a micro graph maintain Invariant 3.c.

Our contraction algorithm is built up from bottom-to-top, i.e., we first describe merges in micro graphs, then mini graphs (and mini boundary graphs) and edge contractions in G (and merges in F). To uphold the responsibilities of micro graphs we implement a variant of the forbidden-vertex graph data structure (Lemma 4) for micro graphs, summed up in the following lemma. To uphold Invariant 4 we are not allowed to merge a vertex v'' to a vertex u'' in a micro graph $P_{i,j}$ if $v'' \in \Delta P_{i,j}$ and $u'' \notin \Delta P_{i,j}$, which we formulate explicitly.

▶ **Lemma 14.** *For all micro graphs $P_{i,j}$ we can provide free assignment merges for each $P_{i,j}$ such that no edges $\{u'', v''\}$ exists that should be managed by F or F_i. If such an edge would occur due to the merge, it is not inserted to $P_{i,j}$ and instead returned. Computing any number of such merges among all micro graphs can be done in $O(n)$ total time. The operation upholds the micro graph-responsibility. Merging a vertex v'' to a vertex u'' is not allowed if $v'' \notin \Delta P_{i,j}$ and $u'' \in \Delta P_{i,j}$. All other (planar preserving) merges are allowed.*

We first note, whenever we call the merge operation of Lemma 14 for a micro graph $P_{i,j}$ in the next paragraphs, the operation returns edges $\{u'', v''\}$ that should be managed by F_i or F, but not $P_{i,j}$. We then translate $\{u'', v''\}$ to $\{u', v'\}$ with u' and v' the respective mini labels of u'' and v''. If the edge $\{u', v'\}$ should be managed by F_i, we insert it to F_i. Returned edges that should not be managed by F_i are instead returned after the merge operation in P_i is executed. This upholds Invariant 3 (restricted to micro and mini graphs). To uphold Invariant 1 (for mini graphs) we check, after any call to a merge of a vertex v'' to u'' in a micro graph $P_{i,j}$ if the degree of u'' changed from 0 to non-zero or vice-versa. If it does, we must possibly update the mapping $\Phi_i^{>0}(u')$ to either include the tuple (j, u'') or remove it, with u' the mini label of u''. Note that this is only done in the case that u' is a boundary vertex, as otherwise no mapping $\Phi_i^{>0}(u')$ exists.

Let $u', v' \in V(P_i)$ be two vertices identified by their external mini label. To provide a merge of u' and v' we distinguish between three cases: (M1) $u', v' \notin \delta P_i$, (M2) $u' \in \delta P_i$ and $v' \notin \delta P_i$ and (M3) $u', v' \in \delta P_i$. In Case M1 we determine the micro graph $P_{i,j}$ that contains micro labels u'' and v'' of u' and v', respectively, via the static mappings $\phi_i(u') = (j, u'')$ and $\phi_i(v') = (j, v'')$ as per Invariant 2. In $P_{i,j}$ we merge v'' to u'' exactly if v' should be merged to u', and otherwise merge u'' to v'' (Lemma 14). By this congruent choice of merge we uphold Invariant 2.c without having to modify any mappings. Since the merged vertex

remains a non-boundary vertex, Invariant 4 is guaranteed. This concludes all responsibilities of merges in mini graphs. All operations take constant time. Note that all merges of Case M1 are free-assignment merges.

Denote with $u^* = \texttt{internal}[u']$ and $v^* = \texttt{internal}[v']$ the internal mini labels of v' and u' respectively. For Case M2 we are forced to merge v^* to u^* to uphold Invariant 4, i.e., internally this merge is not a free-assignment merge. To execute the merge we determine the micro graph $P_{i,j}$ containing the micro label v'' of v^* via $\phi_i(v^*) = (j, v'')$. In $P_{i,j}$ we merge v'' to u'' (Lemma 14) with u'' the micro label of u^* in $P_{i,j}$.

Note that merging u'' to v'' is not allowed. We determine u'' via an operation we call *micro-label search procedure*, which searches for u'' by iterating over all $x'' \in N(v'') \cap \Delta P_{i,j}$ (the neighbors of v'' that are duplicates) and testing if $\Phi_{i,j}^{-1}(x'') = u^*$. If this is the case, we have found the duplicate $u'' := x''$ of u^*. Note that this operation can fail, as u'' and v'' are not guaranteed to be adjacent. In this *special case*, we instead iterate over all $x'' \in \Delta P_{i,j}$. A key characteristic to get a good runtime is that the special case only occurs if the edges $\{u, v\}$ exists in F, with u and v the global labels of u' and v', respectively, which allows us to upper bound the number of encountered special cases by $|E(F)| = O(n/\log^2 n)$ times.

Once the merge is executed in $P_{i,j}$ we must possibly update the mappings that translate between the internal and external mini labels. Recall that the mappings $\texttt{internal}$ and $\texttt{external}$ are only available for vertices of $\delta P_i \cup \Delta P_i$. In the case that $v' \in \delta P_i \cup \Delta P_i$ we are able to provide a free assignment merge as follows: if the request was to merge u' to v', we set $\texttt{internal}[v'] = u^*$ and $\texttt{external}[u^*] = v'$. Otherwise, no update is necessary. If $v' \notin \delta P_i \cup \Delta P_i$ we are not able to provide a free assignment merge, instead we are forced to merge v' to u'. We refer to this situation as the *M2 special case*. If the merge was called with the request to merge u' to v', and we are in this M2 special case, the operation is not allowed. In our use case this case never arises. Intuitively, constraints (e.g., Invariant 4) that force us to contract $\{u, v\}$ by merging v to u either "line up" with being able (or forced) to merge of v' to u' in P_i, with v' and u' the mini labels of v and u in P_i, respectively, or if they do not line up, we make use of the internal/external mappings.

Finally, we consider Case M3. In this case both u' and v' are boundary vertices with $u^* = \texttt{internal}[u']$ and $v^* = \texttt{internal}[v']$ being the internal mini labels of v' and u' respectively. As sketched in Section 2, our intuition for merging v' to u' is that we first merge all v'' to u'' in all micro graphs $P_{i,j}$ that contain both a duplicate u'' of u^* and v'' of v^*. Secondly, for all micro graphs $P_{i,j}$ that contain only a duplicate v'' of v^*, but not of u^*, we update the mappings $\Phi_{i,j}^{-1}(v'') := u^*$ and insert (i, v'') to $\Phi_i(u^*)$. Finally, we merge v^* to u^* in F_i. To describe the realization technically we introduce some additional notation. Denote with $Z_i^{u^* \cap v^*}$ the set of all triples (j, u'', v'') with $(j, u'') \in \Phi_i(u^*)$ and $(j, v'') \in \Phi_i(v^*)$, with $Z_i^{u^* \setminus v^*}$ the set of all tuples (j, u'') with $(j, u'') \in \Phi_i(u^*)$ such that no tuple $(j, \cdot)$ is contained in $\Phi_i(v^*)$, and with $Z_i^{u^* \oplus v^*}$ all tuples $(j, u'') \in \Phi_i(u^*)$ together with all tuples $(j', v'') \in \Phi_i(v^*)$ for which it holds that no tuple $(j', \cdot)$ exists in $\Phi_i(u^*)$. To execute a merge of v^* to u^* in P_i, first iterate over all triples $(j, u'', v'') \in Z_i^{u^* \cap v^*}$ and merge v'' to u'' in $P_{i,j}$, then iterate over all tuples $(j, v'') \in Z_i^{v^* \setminus u^*}$ and update all mappings $\Phi_{i,j}^{-1}(v'') := u^*$. Finally, set $\Phi_i(u^*) := Z_i^{u^* \oplus v^*}$ and merge v^* to u^* in F_i. In the proof we show how these sets occur (intuitively) "naturally" via merges in the graph H_i, which manages Φ_i.

Combining Cases M1, M2 and M3 we show the following lemma.

▶ **Lemma 15.** *All vertex merges in all mini graphs P_i are processed in $O(n)$ time and uphold their responsibilities. Edges that would occur due to a merge in P_i, but should be managed by F are returned. All merges excluding the M2 special case are free assignment merges.*

Contracting edges $\{u, v\}$ in G effectively works exactly as the vertex merges in mini graphs P_i with the exception that we do not need to maintain the translation between internal and external labels, and we do not need to provide free assignment merges for any case. We again distinguish between three cases: (G1) $u, v \notin \delta G$, (G2) $u \in \delta G$ and $v \notin \delta G$ and (G3) $u, v \in \delta G$. For Case G2 we employ a procedure we call *mini-label search procedure*, analogous to the micro-label search procedure for Case M2.

▶ **Lemma 16.** *After $O(n)$ initialization time, any number of edge contractions in G can be computed in $O(n)$ time and uphold the graph responsibility.*

We additionally provide constant time degree queries. Intuitively, we store the degree for boundary vertices (in G and each P_i) concretely, while for all other vertices Theorem 8 provides us with a degree query.

▶ **Lemma 17.** *After $O(n)$ initialization time the degree of any $u \in V$ can be queried in constant time.*

Using the same data structures we use for edge contractions, we can process any number of vertex deletions in $O(n)$ time. To delete a vertex u we delete all mini labels u' of u and all micro labels u'' of all u'. This mostly works analogously to the contraction algorithm.

▶ **Lemma 18.** *Any number of vertex deletions in G can be processed in $O(n)$ time and uphold the graph responsibility.*

We are now able to proof Theorem 1.

▶ **Theorem 1.** *Let $\mathcal{H}(n)$ be the entropy of encoding a planar graph with n vertices and G an unlabeled simple n-vertex planar graph. There exists an encoding of G that provides induced-minor operations (i.e., vertex deletions and edge contractions) with these properties: The encoding requires $O(n)$ time to execute any number of induced-minor operations and provides neighborhood and degree operations in constant time (per element output). The encoding requires $\mathcal{H}(n) + o(n)$ bits can be initialized in $O(n)$ time and $O(n)$ bits.*

Proof. Construct the dynamic encoding due to Corollary 12. Lemma 13 gives us the desired neighborhood operation and Lemma 17 the desired degree operation, Lemma 16 the desired contraction operation and Lemma 18 the desired vertex deletions. ◀

Using hash tables to implement the mappings Φ, $\Phi^{>0}$, Φ_i and $\Phi_i^{>0}$ we are able to provide expected constant time adjacency queries and is able to process any number of edge deletions in $O(n)$ expected time. Holm et al. used the same argument of replacing a mapping data structure with a hash table to show Lemma 5.15 in their work [15].

▶ **Corollary 2.** *The encoding of Theorem 1 can be extended to provide expected $O(1)$ time adjacency queries and process any number of minor operations in expected $O(n)$ time.*

References

1 Joyce Bacic, Saeed Mehrabi, and Michiel Smid. Shortest Beer Path Queries in Outerplanar Graphs. In *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*, volume 212 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 62:1–62:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ISAAC.2021.62.

2 Daniel K. Blandford, Guy E. Blelloch, and Ian A. Kash. Compact representations of separable graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 679–688, USA, 2003. Society for Industrial and Applied Mathematics. doi:10.5555/644108.644219.

3 Guy E. Blelloch and Arash Farzan. Succinct representations of separable graphs. In Amihood Amir and Laxmi Parida, editors, *Combinatorial Pattern Matching*, pages 138–150. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-13509-5_13.

4 Gerth Stølting Brodal and Rolf Fagerberg. Dynamic representations of sparse graphs. In *In Proc. 6th International Workshop on Algorithms and Data Structures (WADS 99)*, pages 342–351. Springer-Verlag, 1999. doi:10.1007/3-540-48447-7_34.

5 Yi-Ting Chiang, Ching-Chi Lin, and Hsueh-I Lu. Orderly spanning trees with applications to graph encoding and graph drawing. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 506–515. Society for Industrial and Applied Mathematics, 2001. doi:10.5555/365411.365518.

6 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

7 Amr Elmasry, Torben Hagerup, and Frank Kammer. Space-efficient Basic Graph Algorithms. In *32nd International Symposium on Theoretical Aspects of Computer Science (STACS 2015)*, volume 30 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 288–301. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.288.

8 Amr Elmasry and Frank Kammer. Space-efficient plane-sweep algorithms. In *27th International Symposium on Algorithms and Computation, ISAAC 2016*, volume 64 of *LIPIcs*, pages 30:1–30:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ISAAC.2016.30.

9 Greg N. Federickson. Fast algorithms for shortest paths in planar graphs, with applications. *SIAM Journal on Computing*, 16(6):1004–1022, 1987. doi:10.1137/0216064.

10 Volodymyr Floreskul, Konstantin Tretyakov, and Marlon Dumas. Memory-efficient fast shortest path estimation in large social networks. *Proceedings of the International AAAI Conference on Web and Social Media*, 8:91–100, 2014. doi:10.1609/icwsm.v8i1.14532.

11 Michael T. Goodrich. Planar separators and parallel polygon triangulation. *J. Comput. Syst. Sci.*, 51(3):374–389, 1995. doi:10.1006/jcss.1995.1076.

12 Torben Hagerup. Space-efficient DFS and applications to connectivity problems: Simpler, leaner, faster. *Algorithmica*, 82(4):1033–1056, 2020. doi:10.1007/s00453-019-00629-x.

13 Xin He, Ming-Yang Kao, and Hsueh-I Lu. A fast general methodology for information-theoretically optimal encodings of graphs. *SIAM Journal on Computing*, 30(3):838–846, 2000. doi:10.1137/S0097539799359117.

14 Klaus Heeger, Anne-Sophie Himmel, Frank Kammer, Rolf Niedermeier, Malte Renken, and Andrej Sajenko. Multistage graph problems on a global budget. *Theoretical Computer Science*, 868:46–64, 2021. doi:10.1016/j.tcs.2021.04.002.

15 Jacob Holm, Giuseppe F. Italiano, Adam Karczmarz, Jakub Lacki, Eva Rotenberg, and Piotr Sankowski. Contracting a Planar Graph Efficiently. In *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ESA.2017.50.

16 Jacob Holm and Eva Rotenberg. Good r-Divisions Imply Optimal Amortized Decremental Biconnectivity. In *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*, volume 187 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.STACS.2021.42.

17 Frank Kammer and Johannes Meintrup. Space-Efficient Graph Coarsening with Applications to Succinct Planar Encodings. In *33rd International Symposium on Algorithms and Computation (ISAAC 2022)*, volume 248 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 62:1–62:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ISAAC.2022.62.

18 Frank Kammer and Johannes Meintrup. Succinct planar encoding with minor operations, 2023. arXiv:2301.10564.

19 Frank Kammer, Johannes Meintrup, and Andrej Sajenko. Space-efficient vertex separators for treewidth. *Algorithmica*, 84(9):2414–2461, 2022. doi:10.1007/s00453-022-00967-3.

20 David R. Karger. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 21–30. Society for Industrial and Applied Mathematics, 1993. doi:10.5555/313559.313605.

21 Kenneth Keeler and Jeffery Westbrook. Short encodings of planar graphs and maps. *Discrete Applied Mathematics*, 58(3):239–252, 1995. doi:10.1016/0166-218X(93)E0150-W.

22 Philip N. Klein and Shay Mozes. *Optimization algorithms for planar graphs*. planarity.org, 2017. URL: http://planarity.org.

23 Philip N. Klein, Shay Mozes, and Christian Sommer. Structured recursive separator decompositions for planar graphs in linear time. In *STOC '13: Proceedings of the forty-fifth annual ACM symposium on Theory of Computing*. Association for Computing Machinery, 2013. doi:10.1145/2488608.2488672.

24 J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, 2001. doi:10.1137/S0097539799364092.

25 Rajeev Raman, Venkatesh Raman, and Srinivasa Rao Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Trans. Algorithms*, 3(4):43–es, November 2007. doi:10.1145/1290672.1290680.

26 Ben Strasser, Dorothea Wagner, and Tim Zeitz. Space-Efficient, Fast and Exact Routing in Time-Dependent Road Networks. In *28th Annual European Symposium on Algorithms (ESA 2020)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 81:1–81:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ESA.2020.81.

27 Hisao Tamaki. Positive-instance driven dynamic programming for treewidth. *J. Comb. Optim.*, 37(4):1283–1311, 2019. doi:10.1007/s10878-018-0353-z.

28 György Turán. On the succinct representation of graphs. *Discret. Appl. Math.*, 8(3):289–294, 1984. doi:10.1016/0166-218X(84)90126-4.