Algorithmica
DOI 10.1007/s00453-010-9452-7

Succinct Representation of Labeled Graphs

Jérémy Barbay · Luca Castelli Aleardi · Meng He · J. Ian Munro

Received: 19 January 2009 / Accepted: 2 September 2010

Abstract In many applications, the properties of an object being modeled are stored as labels on vertices or edges of a graph. In this paper, we consider succinct representation of labeled graphs. Our main results are the succinct representations of labeled and multi-labeled graphs (we consider planar triangulations, planar graphs and k-page graphs) to support various label queries efficiently. The additional space cost to store the labels is essentially the information-theoretic minimum. As far as we know, our representations are the first succinct representations of labeled graphs. We also have two preliminary results to achieve the main contribution. First, we design a succinct representation of unlabeled planar triangulations to support the rank/select of edges in ccw (counter clockwise) order in addition to the other operations supported in previous work. Second, we design a succinct representation for a k-page

The preliminary version of this paper was published in Proceedings of the 18th International Symposium on Algorithms and Computation (ISAAC 2007) [1]. This work was supported by NSERC of Canada, the Canada Research Chairs program, the ERC under the agreement "ERC StG 208471—ExploreMap". The work was done when the first author was in Cheriton School of Computer Science, University of Waterloo, Canada, and part of the second author's work was done during his visit to the Computer Science Department of Université Libre de Bruxelles, Belgium.

J. Barbay
Department of Computer Science (DCC), University of Chile, Santiago, Chile
e-mail: jeremy.barbay@dcc.uchile.cl

L. Castelli Aleardi
LIX, Ecole Polytechnique, Palaiseau, France
e-mail: amturing@lix.polytechnique.fr

M. He (✉) · J.I. Munro
Cheriton School of Computer Science, University of Waterloo, 200 University Avenue West, Waterloo, Ontario, N2L 3G1 Canada
e-mail: mhe@uwaterloo.ca

J.I. Munro
e-mail: imunro@uwaterloo.ca

Published online: 16 September 2010

graph when k is large to support various navigational operations more efficiently. In particular, we can test the adjacency of two vertices in $O(\lg k)$ time, while previous work uses $O(k)$ time.

Keywords Succinct data structures · Data structures · Graph · Planar graph · Planar triangulation · Book embedding · k-page graph

1 Introduction

Graphs are fundamental combinatorial objects in mathematics and in computer science. They are widely used to represent various types of data, such as the link structure of the web, geographic maps, and surface meshes in computer graphics. As modern applications often process large graphs, the problem of designing space-efficient data structures to represent graphs has attracted a great deal of attention. In particular the idea of succinct data structures has been applied to various classes of graphs [6–10, 18, 22].

Previous work focused on succinct graph representations which efficiently support testing the adjacency between two vertices and listing the edges incident with a vertex [7, 8, 22]. However, in many applications, such connectivity information is associated with labels on the edges or vertices of the graph, and the space required to encode those labels dominates the space used to encode the connectivity information, even when the encoding of the labels is compressed [17]. For example, when surface meshes are associated with properties such as color and texture information, more bits per vertex are required to encode those labels than to encode the graph itself. We address this problem by designing succinct representations of *labeled graphs*, where labels from alphabet $[\sigma]$[1] are associated with edges or vertices. These representations support label-based connectivity queries efficiently, such as retrieving the neighbors associated with a given label. Our results are under the word RAM model with word size $\Theta(\lg n)$ bits. We assume that all the graphs are simple graphs.

We investigate three important classes of graphs: planar triangulations, planar graphs and k-page graphs. Planar graphs, and in particular planar triangulations, correspond to the connectivity information underlying surface meshes. Triangulated meshes are one of the most fundamental representations for geometric objects: in computational geometry they are one natural way to represent surface models, and in computer graphics triangles are the basic geometric primitive for efficient rendering. k-page graphs have applications in several areas, such as sorting with parallel stacks [28], fault-tolerant processor arrays [26] and VLSI (very large scale integration) design [11].

The rest of the paper is organized as follows. Section 2 gives a brief review of previous work. We describe existing results that we use and/or improve upon in Sect. 3. In Sect. 4, we present succinct indexes for triangulated planar graphs with labels associated with their vertices or edges, and use them to design succinct indexes for

[1] We use $[\sigma]$ to denote the set $\{1, 2, \ldots, \sigma\}$ of references to arbitrary labels, as indeed the alphabet of labels.

multi-labeled general planar graphs. To achieve these results, we describe a succinct representation of unlabeled planar triangulations which supports the rank/select of edges in ccw (counter clockwise) order in addition to the other operations supported in previous work. We present a succinct encoding for k-page graphs with labels associated with their edges in Sect. 5. To achieve this result, we design a succinct representation for a k-page graph when k is large, which supports various navigational operations more efficiently. We conclude with a discussion of our results in Sect. 6.

2 Previous Work

Here we briefly review related work on succinct unlabeled graphs. As graphs in practice often have particular combinatorial properties, researchers usually exploit these properties to design succinct representations.

Jacobson [18] was the first to propose a succinct representation of planar graphs. His approach is based on the concept of *book embedding* by Bernhart and Kainen [5]. A *k-page embedding* is a topological embedding of a graph with the vertices along the spine and edges distributed across k pages, each of which is an outerplanar graph. The minimum number of pages, k, for a particular graph has been called the *pagenumber* or *book thickness*. Jacobson showed how to represent a k-page graph using $O(kn)$ bits to support adjacency tests in $O(\lg n)$ bit probes,[2] and listing the neighbors of a vertex x in $O(d(x)\lg n + k)$ bit probes, where $d(x)$ is the degree of x.

Munro and Raman [22] improved Jacobson's results under the word RAM model by showing how to represent a graph using $2kn + 2m + o(kn + m)$ bits to support adjacency tests and the computation of the degree of a vertex in $O(k)$ time, and the listing of all the neighbors of a given vertex in $O(d + k)$ time. Gavoille and Hanusse [14] proposed a different tradeoff. They proposed an encoding in $2(m + i)\lg k + 4(m + i) + o(km)$ bits, where i is the number of isolated vertices, to support the adjacency test in $O(k)$ time. As any planar graph can be embedded in at most 4 pages [30], these results can be applied to planar graphs directly. In particular, a planar graph can be represented using $8n + 2m + o(n)$ bits to support adjacency tests and the computation of the degree of a vertex in $O(1)$ time, and the listing of all the neighbors of a given vertex x in $O(d(x))$ time (i.e. constant time per neighbor) [22].

A different line of research based on the spanning trees of planar graphs was taken by Chuang *et al.* [10]. They designed a succinct representation of planar graphs of n vertices and m edges in $2m + (5 + \epsilon)n + o(m + n)$ bits, for any constant $\epsilon > 0$, to support the operations on planar graphs in asymptotically the same amount of time as the approach described in the previous paragraph. Chiang *et al.* [9] further reduced the space cost to $2m + 2n + o(m + n)$ bits. When a planar graph is triangulated, Chuang *et al.* [10] showed how to represent it using $2m + n + o(m + n)$ bits. Yamanaka and Nakano [29] further showed how to represent a planar triangulation using $2m + o(m)$ bits to provide the same support for operations.

[2] We use $\log_2 x$ to denote the logarithmic base 2 and $\lg x$ to denote $\lceil \log_2 x \rceil$. Occasionally this matters.

Based on a partition algorithm, Castelli Aleardi *et al.* [7] proposed a succinct representation of planar triangulations with a boundary. Their data structure uses 2.175 bits per triangle to support various operations efficiently. Castelli Aleardi *et al.* [8] further extended this approach to design succinct representations of 3-connected planar graphs and triangulations using 2 bits per edge and 1.62 bits per triangle respectively, which asymptotically match the respective entropy of these two types of graphs.

Blandford *et al.* [6] considered the problem of representing graphs with small separators (the graph separator considered in their main result is a *vertex separator*, i.e. a set of vertices whose removal separates the graph into two approximately equally sized parts). This is useful because many graphs in practice, including planar graphs [20], have small separators. They designed a succinct representation using $O(n)$ bits that supports adjacency tests and the computation of the degree of a vertex in $O(1)$ time, and the listing of all the neighbors of a given vertex x in $O(d(x))$ time.

Finally, Farzan and Munro [13] considered general directed graphs, and showed how to represent a directed graph using $(1+\epsilon)\left(\lg \binom{n^2}{m}\right)$ bits, for any arbitrarily small constant $\epsilon > 0$, to support adjacency tests and the computation of the degree of a vertex in $O(1)$ time, as well as the listing of the neighbors of a vertex x in $O(d(x))$ time. This space cost is within a multiplicative $1+\epsilon$ factor of the information-theoretic lower bound. They also showed that it is impossible to represent a directed graph using $\lg \binom{n^2}{m} + o\left(\lg \binom{n^2}{m}\right)$ bits to provide the same support for operations, by proving a lower bound for this problem. Similar upper and lower bounds can be proved for undirected graphs.

3 Preliminaries

3.1 Bit Vectors

A key structure for the design of many succinct data structures, and for the research work in this paper, is a bit vector B of length n supporting *rank* and *select* operations. The positions in B are numbered $1, 2, \ldots, n$. We consider the following operations for $\alpha \in \{0, 1\}$:

- $\texttt{rank}_B(\alpha, x)$, the number of occurrences of α in $B[1..x]$;
- $\texttt{select}_B(\alpha, r)$, the position of the rth occurrence of α in B.

We omit the subscript B when it is clear from the context. Lemma 1 summarizes previous results on succinct representations of bit vectors, in which part (a) is from Jacobson [18] and Clark and Munro [12], while part (b) is from Raman *et al.* [25].

Lemma 1 *A bit vector B of length n with v 1s can be represented using either*: (a) $n + o(n \lg\lg n / \lg n)$ *bits*, *or* (b) $\lg \binom{n}{v} + O(n \lg\lg n / \lg n)$ *bits*, *to support the access to each bit*, `rank` *and* `select` *in* $O(1)$ *time*.

3.2 Balanced Parentheses

Another structure we use is a balanced parenthesis sequence S of length $2n$, where there are n opening parentheses and n closing parentheses. The following operations are considered:

- $\texttt{rank\_open}_S(i)$, the number of opening parenthesis in $S[1..i]$;
- $\texttt{rank\_close}_S(i)$, the number of closing parenthesis in $S[1..i]$;
- $\texttt{select\_open}_S(i)$, the position of the ith opening parenthesis in S;
- $\texttt{select\_close}_S(i)$, the position of the ith closing parenthesis in S;
- $\texttt{find\_open}_S(i)$, the matching opening parenthesis for the closing parenthesis at position i;
- $\texttt{find\_close}_S(i)$, the matching closing parenthesis for the opening parenthesis at position i;
- $\texttt{excess}_S(i)$, the number of opening parentheses minus the number of closing parentheses in $S[1..i]$;
- $\texttt{enclose}_S(i)$, the closest enclosing (matching parenthesis) pair of a given matching parenthesis pair whose opening parenthesis is at position i.

The subscript S is omitted when it is clear from the context.

Munro and Raman [22] considered the problem of representing a balanced parenthesis sequence succinctly, and their result is:

Lemma 2 [22] *A sequence of balanced parentheses S of length $2n$ can be represented using $2n + o(n)$ bits to support the operations* `rank_open`, `rank_close`, `select_open`, `select_close`, `find_close`, `find_open`, `excess` *and* `enclose` *in $O(1)$ time.*

3.3 Multiple Parentheses

Chuang *et al.* [10] proposed the succinct representation of *multiple parentheses*, a string of $O(1)$ types of parentheses that may be unbalanced. Thus a multiple parenthesis sequence of p types of parentheses is a sequence over the alphabet $\{'(_1', ')_1', '(_2', ')_2', \ldots, '(_p', ')_p'\}$. We call $'(_i'$ and $')_i'$ *type-i opening parenthesis* and *type-i closing parenthesis*, respectively. This is a generalization of balanced parentheses. The operations considered are:

- $\texttt{m\_rank}_S(\alpha, i)$, the number of occurrences of parentheses α in $S[1..i]$;
- $\texttt{m\_select}_S(\alpha, i)$, the position of the ith occurrence of parenthesis α;
- $\texttt{m\_first}_S(\alpha, i)$, the position of the first occurrence of parenthesis α after the ith position in S;
- $\texttt{m\_last}_S(\alpha, i)$, the position of the last occurrence of parenthesis α before the ith position in S;
- $\texttt{m\_match}_S(i)$, the position of the parenthesis matching $S[i]$;
- $\texttt{m\_enclose}_S(k, i_1, i_2)$, the position of the closest matching parenthesis pair of type k which encloses $S[i_1]$ and $S[i_2]$.

We omit the subscript S when it is clear from the context. Chuang *et al.* [10] showed how to support the above operations:

Lemma 3 [10] *Consider a string S of $O(1)$ types of parentheses that is stored explicitly. There is an auxiliary data structure using $o(|S|)$ bits that supports the operations listed above in $O(1)$ time.*

We show how to improve this result in Lemma 12, and propose an encoding for the case in which the number of types of parentheses is non-constant in Theorem 7.

3.4 Succinct Indexes for Strings and Binary Relations

Barbay *et al.* [2, 3] showed how to achieve data abstraction in succinct data structures by designing *succinct indexes*. Given an abstract data type (ADT) to access the given data, the goal is to design auxiliary data structures (i.e. succinct indexes) that occupy asymptotically less space than the information-theoretic lower bound on the space required to encode the given data, and support an extended set of operations using the basic operators defined in the ADT. They considered a string S of length n over an alphabet of arbitrary size σ that support the following operations:

- $\texttt{string\_rank}_S(\alpha, x)$, the number of occurrences of α in $S[1..x]$;
- $\texttt{string\_select}_S(\alpha, r)$, the position of the rth occurrence of α in the string;
- $\texttt{string\_access}_S(x)$, the character at position x in the string.

The subscript S is omitted when it is clear from the context. They defined the interface of the ADT of a string through the operator `string_access`, and designed a succinct index of $n \cdot o(\lg \sigma) + O(n)$ bits that provides efficient support for `string_rank` and `string_select` using `string_access` to access the string.

They also extended the problem to n objects where each object can be associated with a subset of labels from $[\sigma]$, this association being defined by a binary relation R of t pairs from $[n] \times [\sigma]$. The operations include:

- $\texttt{object\_select}_R(x, i)$, the ith label associated with x in lexicographic order, and $+\infty$ if no such label exists;
- $\texttt{label\_rank}_R(\alpha, x)$, the number of objects labeled α up to (and including) x;
- $\texttt{label\_select}_R(\alpha, r)$, the position of the rth object labeled α;
- $\texttt{label\_access}_R(x, \alpha)$, whether object x is associated with label α.

The subscript R is omitted when it is clear from the context. They defined the interface of the ADT of a binary relation through the operator `object_select` and designed a succinct index of $t \cdot o(\lg \sigma)$ bits to support other operators efficiently. As we use this result extensively in our paper, we summarize it in the following lemma (they assumed that each object is associated with at least one label):

Lemma 4 [2] *Given support for* `object_select` *in* $f(n, \sigma, t)$ *time on a binary relation* R *formed by* t *pairs from an object set* $[n]$ *and a label set* $[\sigma]$, *there is a succinct index using* $t \cdot o(\lg \sigma) + O(t)$ *bits that supports* `label_rank` *in* $O((\lg\lg\lg\sigma)^2(f(n, \sigma, t) + \lg\lg\sigma))$ *time*, `label_select` *in* $O(\lg\lg\lg\sigma(f(n, \sigma, t) + \lg\lg\sigma))$ *time*, *and* `label_access` *in* $O(\lg\lg\lg\sigma \cdot f(n, \sigma, t) + \lg\lg\sigma)$ *time*.

Barbay *et al.* [2] further showed how to use succinct indexes to represent a binary relation using space close to the information-theoretic minimum (i.e. $\lg \binom{n\sigma}{t}$ bits):

Lemma 5 [2] *A binary relation* R *formed by* t *pairs from an object set* $[n]$ *and a label set* $[\sigma]$ *can be represented using* $\lg \binom{n\sigma}{t} + t \cdot o(\lg \sigma) + O(t)$ *bits to support* `object_select` *in* $O(1)$ *time*, `label_rank` *in* $O(\lg\lg\sigma(\lg\lg\lg\sigma)^2)$ *time*,

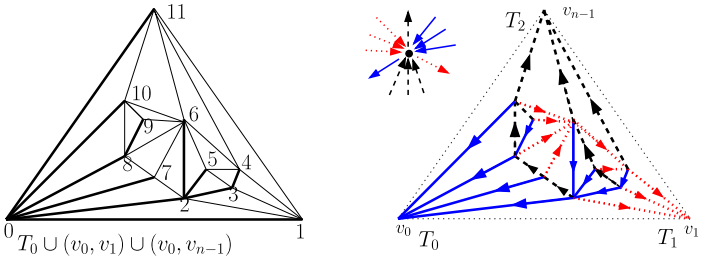

Fig. 1 A triangulated planar graph of 12 vertices with its canonical spanning tree $\overline{T_0}$ (on the *left*). On the *right*, it shows the triangulation induced with a realizer, as well as the local condition

`label_select` *in* $O(\lg\lg\sigma \lg\lg\lg\sigma)$ *time*, *and* `label_access` *in* $O(\lg\lg\sigma)$ *time*.

They showed that the techniques used to prove Lemmas 4 and 5 also apply to the more general case in which each object is associated with zero or more labels. In this more general case, they demonstrated that the space costs in both lemmas are only increased by $O(n)$ bits, while the support for the operations remains the same.

3.5 Realizers and Planar Triangulations

Our general approach, in much of this paper, is based on the idea of realizers (also known as Schnyder trees or Schnyder woods) of planar triangulations (see Fig. 1 for an example).

Definition 1 [27] A **realizer** of a planar triangulation $\mathcal{T}$ is a partition of the set of the internal edges into three sets T_0, T_1 and T_2 of directed edges, such that for each internal vertex v the following conditions hold:

- v has exactly one outgoing edge in each of the three sets T_0, T_1 and T_2;
- **local condition:** the edges incident with v in counterclockwise (ccw) order are: one outgoing edge in T_0, zero or more incoming edges in T_2, one outgoing edge in T_1, zero or more incoming edges in T_0, one outgoing edge in T_2, and finally zero or more incoming edges in T_1.

From local conditions of the definition, it is possible to state a more global structural property (characterizing very finely the notion of planarity for triangulations) which is expressed in terms of a spanning condition as below (we use it extensively in Sect. 4):

Lemma 6 [27] *Consider a planar triangulation $\mathcal{T}$ of n vertices, with exterior face (v_0, v_1, v_{n-1}). Then $\mathcal{T}$ always admits a realizer $R = (T_0, T_1, T_2)$ and each set of edges in T_i is a spanning tree of all internal vertices. More precisely*:

- *T_0 is a spanning tree of $\mathcal{T} \setminus \{v_1, v_{n-1}\}$;*
- *T_1 is a spanning tree of $\mathcal{T} \setminus \{v_0, v_{n-1}\}$;*
- *T_2 is a spanning tree of $\mathcal{T} \setminus \{v_0, v_1\}$.*

As we consider undirected planar triangulations, we orient each internal edge when we compute the realizers. If we reverse the direction of each edge in T_i, we get a different set of directed edges. We use T_i^{-1} to denote this set. We also use the following lemma in this paper:

Lemma 7 [27] *If T_0, T_1 and T_2 define a realizer of a planar triangulation $\mathcal{T}$, then for $i \in \{0, 1, 2\}$, there is no directed cycle in the set $T_i \cup T_{i+1}^{-1} \cup T_{i+2}^{-1}$ (indices are modulo 3).*

4 Planar Triangulations

4.1 Three New Traversal Orders on a Planar Triangulation

A key notion in the development of our results is that of three new traversal orders of planar triangulations based on realizers. Let $\mathcal{T}$ be a planar triangulation of n vertices and m edges, with exterior face (v_0, v_1, v_{n-1}). We denote a realizer of $\mathcal{T}$ by (T_0, T_1, T_2) following Definition 1. By Lemma 6, T_0, T_1 and T_2 are three spanning trees of the internal vertices of $\mathcal{T}$, rooted at v_0, v_1 and v_{n-1}, respectively. We add the edges (v_0, v_1) and (v_0, v_{n-1}) to T_0, and call the resulting tree, $\overline{T_0}$, the *canonical spanning tree* of $\mathcal{T}$ [10]. In this section, we identify each vertex by its rank in *canonical ordering*, which is the ccw preorder rank in $\overline{T_0}$ (i.e. vertex i or v_i denotes the ith vertex in canonical ordering). We use (x, y) to indicate the edge between vertices x and y. We will be describing planar graphs/triangulations and their tree decomposition. As an aid to remind the reader of the context, we will tend to use the term *node* when discussing a tree and *vertex* in the context of the full planar graph. For example, we may make such a statement as "vertex x is a leaf node of the canonical spanning tree, $\overline{T_0}$, of graph $\mathcal{T}$".

Definition 2 The **zeroth order**, π_0, is defined on all the vertices of $\mathcal{T}$ and is simply given by the preorder traversal of $\overline{T_0}$ starting at v_0 in *counterclockwise order* (ccw order).

The **first order**, π_1, is defined on the vertices of $\mathcal{T} \setminus v_0$ and corresponds to a traversal of the edges of T_1 as follows. Perform a preorder traversal of the contour of $\overline{T_0}$ in a ccw manner. During this traversal, when visiting a node v, we enumerate consecutively its incident edges $(v, u_1), \ldots, (v, u_i)$ in T_1, where v appears before u_i in π_0. The traversal of the edges of T_1 naturally induces an order on the nodes of T_1: each node (different from v_1) is uniquely associated with its parent edge in T_1.

The **second order**, π_2, is defined on the vertices of $\mathcal{T} \setminus \{v_0, v_1\}$ and can be computed in a similar manner by performing a preorder traversal of T_0 in *clockwise order* (cw order). When visiting the contour of $\overline{T_0}$ in cw order, the edges in T_2 incident with a node v are listed consecutively to induce an order on the nodes of T_2.

Figure 2 gives an example of the above three orders in a planar triangulation. Note that the orders π_1 and π_2 do not correspond to previously studied traversal orders on the trees T_1 and T_2, as they are dependent on $\overline{T_0}$ through π_0. To show that all the internal vertices are listed in π_1 and π_2, it suffices to prove the following lemma.

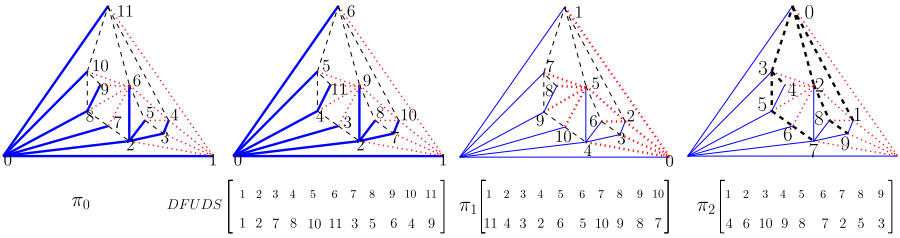

Fig. 2 A planar triangulation induced with one realizer. The three orders π_0, π_1 and π_2, as well as the order induced by a DFUDS traversal [4] of $\overline{T_0}$ are also shown

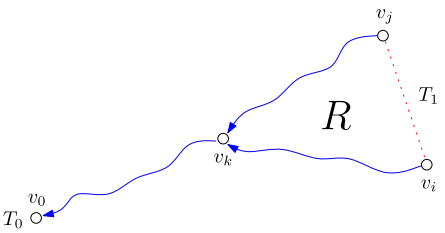

Fig. 3 Region R in the proofs of Lemmas 8 and 10

Lemma 8 *Given an edge (v_i, v_j), where $i < j$, if it is in T_1, then v_i is v_j's parent in T_1. If this edge is in T_2, then v_j is v_i's parent in T_2.*

Proof We consider only the case in which the edge (v_i, v_j) is in T_1; the claim for the case in which (v_i, v_j) is in T_2 can be proved similarly.

As the case in which $i = 1$ is trivial, and there is no edge in T_1 that is incident with v_0 or v_{n-1}, we need only consider the case in which v_i and v_j are internal vertices.

We first prove that v_i is not v_j's ancestor in T_0. Assume to the contrary that v_i is v_j's ancestor in T_0. Recall that the edges in T_0, T_1 and T_2 are oriented toward the parent nodes incident with them. Then there is a directed path from v_j to v_i in T_0. As there is an edge in T_1 between v_i and v_j, there is a directed cycle from v_j to v_i and then back to v_j using edges from the set $T_0 \cup T_1^{-1}$ or the set $T_1^{-1} \cup T_0^{-1}$ (depending on the direction of the edge (v_i, v_j)), which contradicts Lemma 7.

Let v_k be the lowest common ancestor of v_i and v_j in T_0. As $i < j$ and v_i is not v_j's ancestor in T_0, the path from v_i to v_k in T_0, the path from v_j to v_k in T_0 and the edge (v_i, v_j) define a closed region R (see Fig. 3). Assume to the contrary that v_j is v_i's parent in T_1. Then, according to the local condition in Definition 1, the parent of v_j in T_2 is either inside R or on the boundary of R. As there is a directed path from v_j to v_{n-1} and v_{n-1} is neither inside R or on its boundary, we conclude that there exists a vertex v_t either in the path from v_i to v_k in T_0, or in the path from v_j to v_k in T_0, that is v_j's ancestor in T_2. In the former case, there is a directed cycle $v_i, \ldots, v_t, \ldots, v_j, v_i$ consisting of edges in the set $T_0 \cup T_1^{-1} \cup T_2^{-1}$. In the latter case, there is a directed cycle $v_j, \ldots, v_t, \ldots, v_j$ consisting of edges in the set $T_0 \cup T_2^{-1}$. Either of these two observations contradicts Lemma 7. □

The following lemma is crucial, as it puts in correspondence the labels of the neighbors of a vertex with a finite number of substrings.

Lemma 9 *For any vertex x, its children in T_1 listed in ccw order have consecutive ranks in π_1. Similarly, x's children in T_2 listed in cw order have consecutive ranks in π_2. In the case of $\overline{T_0}$, the children of x are listed consecutively by a* DFUDS *(i.e. Depth First Unary Degree Sequence* [4]*) traversal*[3] *of $\overline{T_0}$.*

Proof This lemma follows directly from the local condition and the ccw traversal performed on T_0 to construct π_1 and π_2. To be specific, the edges between x and its children in T_1 are all incoming edges incident with x in T_1, and because of the local condition in Definition 1, they are encountered consecutively when listing the edges incident with x in ccw order (and just before visiting the outgoing edge of x in T_0). A similar argument holds for π_2 and π_0. □

4.2 Representing Planar Triangulations

We consider the following operations on unlabeled planar triangulations:

- `adjacency`(x, y), whether vertices x and y are adjacent;
- `degree`(x), the degree of vertex x;
- `select_neighbor_ccw`(x, y, r), the rth neighbor of vertex x starting from vertex y in ccw order if x and y are adjacent, and ∞ otherwise;
- `rank_neighbor_ccw`(x, y, z), the number of neighbors of vertex x between (and including) the vertices y and z in ccw order if y and z are both neighbors of x, and ∞ otherwise;
- $\Pi_j(i)$, given the rank of a vertex in π_0, returns the rank of the same vertex in π_j ($j \in \{1, 2\}$);
- $\Pi_j^{-1}(i)$, given the rank of a vertex in π_j, returns the rank of the same vertex in π_0 ($j \in \{1, 2\}$).

To represent a planar triangulation $\mathcal{T}$, we compute a realizer (T_0, T_1, T_2) of $\mathcal{T}$ following Lemma 6. We then encode the three trees T_0, T_1 and T_2 using a multiple parenthesis sequence S of $2(m + 1)$ parentheses of three types. S is obtained by performing a preorder traversal of the canonical spanning tree $\overline{T_0} = T_0 \cup (v_0, v_1) \cup (v_0, v_{n-1})$ and using different types of parentheses to describe the edges of $\overline{T_0}$, T_1 and T_2. We use parentheses of the first type, namely $'('$ and $')'$, to encode the tree $\overline{T_0}$, and other types of parentheses, $'['$, $']'$, $'\{'$, $'\}'$, to encode the edges of T_1 and T_2. We use S_0, S_1 and S_2 to denote the subsequences of S that contain all the parentheses of the first, second, and the third types, respectively. We construct S as follows (see Fig. 4 for an example).

[3] In the DFUDS sequence of a tree, a node of degree d is represented by d opening parentheses followed by a closing parenthesis. All the nodes are listed in preorder in the sequence, and an extra opening parenthesis is added to the beginning of the sequence. Each node is numbered by its opening parenthesis in its parent's description, and this number is called the DFUDS rank. The DFUDS traversal visits the nodes of a tree by their DFUDS ranks. Figure 2 also shows the DFUDS ranks of the nodes in $\overline{T_0}$.

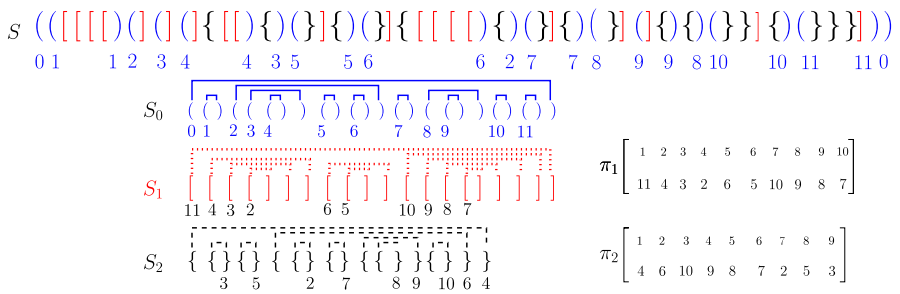

Fig. 4 The multiple parenthesis string encoding of the planar triangulation in Fig. 2

Let $v_0, \ldots, v_{n-1}$ be the ccw preorder of the nodes of $\overline{T_0}$. Then the string S_0 is simply the balanced parenthesis encoding of the tree $\overline{T_0}$ [22]: S_0 can be obtained by performing a ccw preorder traversal of the contour of $\overline{T_0}$, writing down an opening parenthesis when an edge of $\overline{T_0}$ is traversed for the first time, and a closing parenthesis when it is visited for the second time. During the traversal of $\overline{T_0}$, we insert in S a pair of parentheses $'['$ and $']'$ for each edge of T_1, and a pair of parentheses $'\{'$ and $'\}'$ for each edge in T_2. More precisely, when visiting the edges incident with a vertex v_i in ccw order, we insert:

- A $'['$ for each edge (v_i, v_j) in T_1, where $i < j$, before the parenthesis $')'$ corresponding to v_i;
- A $']'$ for each edge (v_i, v_j) in T_1, where $i < j$, after the parenthesis $'('$ corresponding to v_j;
- A $'\}'$ for each edge (v_i, v_j) in T_2, where $i > j$, after the parenthesis $'('$ corresponding to v_i;
- A $'\{'$ for each edge (v_i, v_j) in T_2, where $i > j$, before the parenthesis $')'$ corresponding to v_j.

The relative order of the parentheses $'['$, $']'$, $'\}'$ and $'\{'$ inserted between two consecutive parentheses of the other type, i.e. $'('$ or $')'$, does not matter. For the simplicity of the proofs in this section, we assume that when we insert a parenthesis pair for an edge in T_1 or T_2, we always ensure that the positions of this pair in S either enclose or are enclosed in those for an edge that shares the same parent node in the same tree. There are $n-2$ and $n-3$ edges in T_1 and T_2, respectively, so the subsequences S_1 and S_2 are balanced parenthesis sequences of lengths $2(n-2)$ and $2(n-3)$, respectively. As S_0 is the balanced parenthesis encoding of $\overline{T_0}$, the length of S_0 is $2n$. Therefore, the string S is of length $2n + 2(n-2) + 2(n-3) = 2(3n-5) = 2(m+1)$, consisting of three types of parenthesis.

We first observe some basic properties of the string S. Recall that a vertex v_i can be referred to by its preorder rank in $\overline{T_0}$, and by the position of the matching parenthesis pair of the first type corresponding to it. We use p_i and q_i to denote the positions of the opening and closing parentheses of this pair in S respectively. Let p_f be the position of the opening parenthesis in S corresponding to the first child of v_i in $\overline{T_0}$, and q_l be the position of the closing parenthesis in S corresponding to the last child of v_i in $\overline{T_0}$.

Property 1 The following basic facts hold:

- Two vertices v_i and v_j are adjacent if and only if there is one common incident edge (v_i, v_j) in exactly one of the trees $\overline{T_0}$, T_1 or T_2;
- $p_i < p_f < q_l < q_i$;
- The number of edges incident with v_i and not belonging to the tree $\overline{T_0}$ is $(p_f - p_i - 1) + (q_i - q_l - 1)$;
- If v_i is not a leaf in $\overline{T_0}$, between the occurrences of the $'('$ that correspond to the vertices v_i and v_{i+1} (note that the $'('$ corresponding to v_{i+1} is at position p_f), there is exactly one $']'$. Similarly, there is exactly one $'\{'$ between the $')'$ that corresponds to the vertices v_i and the $')'$ at position q_l.

We now prove the following lemma that is important to our representation:

Lemma 10 *In the process of constructing S, the two parentheses of the second type inserted for each edge in T_1 form a matching parenthesis pair in S. Similarly, the two parentheses of the third type inserted for each edge in T_2 form a matching parenthesis pair in S.*

Proof We prove the lemma for parentheses of the second type; the claim for parentheses of the third type follows similarly.

Recall that for each edge (v_i, v_j) in T_1, where $i < j$, we insert a $'['$ and a $']'$ into S. We first prove that between these two parentheses, the position of parenthesis $'['$ in S is before that of parenthesis $']'$. As the case in which $i = 1$ is trivial, and no edge in T_1 is incident with v_0 or v_{n-1}, we need only consider the case in which v_i and v_j are internal vertices. By the process of constructing S, it suffices to prove that the closing parenthesis $')'$ corresponding to v_i appears before the opening parenthesis $'('$ corresponding to v_j. Assume to the contrary that this is not true. As the parenthesis $'('$ corresponding to v_i appears before the parenthesis $'('$ corresponding to v_j (this is because $i < j$), we conclude that the parenthesis pair corresponding to v_i in S_0 encloses the pair corresponding to v_j. Thus v_i is an ancestor of v_j in T_0. Therefore, there is a directed path from v_j to v_i using edges from the set T_0. By Lemma 8, v_i is v_j's parent in T_1, so the edge (v_i, v_j) is oriented toward v_i. Hence there is a directed cycle $v_j, \ldots, v_i, v_j$ using edges from the set $T_0 \cup T_1^{-1}$, which contradicts Lemma 7.

It now suffices to prove that the two parenthesis pairs of the second type inserted for two different edges in T_1 either do not intersect, or one is enclosed in the other. Let (v_i, v_j) $(i < j)$ and (v_p, v_q) $(p < q)$ be two edges that are not incident with the same node in T_1 (the case in which these two edges are incident with the same node in T_1 is trivial). As shown in the proof of Lemma 8, we have that v_i is not v_j's ancestor in T_0, and that v_p is not v_q's ancestor in T_0. Let v_k be the lowest common ancestor of v_i and v_j in T_0. We define the region R as in the proof of Lemma 8 (see Fig. 3). Without the loss of generality, we assume that $p < i$. There are two cases.

We first consider the case in which v_p is v_i's ancestor. If v_p is also v_j's ancestor, then the parenthesis pair of the first type corresponding to v_p encloses the pairs corresponding to v_i and v_j. By the process we use to construct S, the two parenthesis pairs of the second type inserted for edges (v_i, v_j) and (v_p, v_q) do not intersect. If v_p is not v_j's ancestor, then it is in the path from v_i to v_k in T_0, excluding v_i and v_k.

We observe that the parenthesis $'['$ inserted for (v_p, v_q) is after the one inserted for (v_i, v_j). By the local condition in Definition 1 and the planarity of the graph, v_q is either inside region R, or is in the path from v_j to v_k in T_0. Therefore, $q < j$. Thus the parenthesis $'('$ corresponding to v_q is before the parenthesis $'('$ corresponding to v_j. Hence the parenthesis $']'$ inserted for (v_p, v_q) is before the one inserted for (v_i, v_j). Therefore, the parenthesis pair of the second type inserted for (v_p, v_q) is enclosed in the pair inserted for (v_i, v_j).

We next consider the case in which v_p is not v_i's ancestor. In this case, the parenthesis $')'$ corresponding to v_p appears before that corresponding to v_i. Thus the parenthesis $'['$ inserted for (v_p, v_q) appears before the one inserted for (v_i, v_j). As v_p is outside R, by the planarity of the graph, v_q is either outside R, or on R's boundary. We also observe that v_q cannot be in the path from v_j to v_k in T_0, because otherwise, by the local condition in Definition 1, v_p is either in R or in the path from v_i to v_k in T_0. Therefore, v_q is either before v_i in canonical order, or is a descendant of v_i, or is after v_j in canonical order. In the first two cases, the parenthesis pairs of the second type inserted for (v_p, v_q) and (v_i, v_j) do not intersect. In the last case, the parenthesis pair of the second type inserted for (v_i, v_j) is enclosed by that for (v_p, v_q). □

Observe that S_0 is the balanced parenthesis encoding of the tree $\overline{T_0}$ [22], so if we store S_0 and construct the auxiliary data structures for S_0 as in [9, 21–23], we can support a set of navigational operators on $\overline{T_0}$. S can be represented using Lemma 3 in $2m\lceil\log_2 6\rceil + o(m) = 6m + o(m)$ bits. However, this encoding does not support the computation of an arbitrary word in S_0, so we cannot navigate in the tree $\overline{T_0}$ without storing S_0 explicitly, which will cost essentially 2 additional bits per node. To reduce this space redundancy, and to decrease the term $2m\lceil\log_2 6\rceil$ to $2m\log_2 6 + o(m)$, we have the following lemma.

Lemma 11 *The string S can be stored in $2m\log_2 6 + o(m)$ bits to support the operators listed in Sect. 3.3 in $O(1)$ time, as well as the computation of an arbitrary word or $\Theta(\lg(n))$ bits of the balanced parenthesis sequence of $\overline{T_0}$ in $O(1)$ time.*

Proof We construct a bit vector B_1 of $2(m+1)$ bits, so that $B_1[i] = 1$ iff $S[i] = '('$ or $S[i] = ')'$. We construct another bit vector B_2 of $2m + 2 - 2n$ bits for the 0s in B_2 (recall that there are $2n$ parentheses of the first type), so that $B_2[i] = 1$ iff the parenthesis corresponds to the ith 0 in B_1 is either $'['$ or $']'$. We store B_1 and B_2 using Part (b) of Lemma 1 to support rank/select operations on them. The space cost of storing B_1 and B_2 is thus $\lg\binom{2m+2}{n} + o(m) + \lg\binom{2m+2-2n}{2n-4} + o(m)$. To analyze the above space cost, we use the equality $\log_2 n! = n\log_2 n - n\log_2 e + \frac{1}{2}\log_2 n + O(1)$ [15, Sect. 4.6.4]. We have (note that $m = 3n - 6$):

$$
\begin{aligned}
&\log_2\binom{2m+2}{2n} + \log_2\binom{2m+2-2n}{2n-4} \\
&= \log_2\binom{6n-10}{2n} + \log_2\binom{4n-10}{2n-4} \\
&= \log_2\left(\frac{(6n-10)!}{(2n)!(4n-10)!} \times \frac{(4n-10)!}{(2n-4)!(2n-6)!}\right)
\end{aligned}
$$

$$
\begin{aligned}
&= \log_2 \frac{(6n-10)!}{(2n)!(2n-4)!(2n-6)!} \\
&< \log_2 \frac{(6n)!}{((2n)!)^3} \\
&= \log_2 (6n)! - 3\log_2 (2n)! \\
&= 6n\log_2(6n) - 6n\log_2 e + \frac{1}{2}\log_2(6n) \\
&\quad - 3\left(2n\log_2(2n) - 2n\log_2 e + \frac{1}{2}\log_2(2n)\right) + O(1) \\
&= 6n\log_2 3 - \log_2 n + O(1) \\
&< 6n\log_2 3 + O(1) \\
&= 2m\log_2 3 + O(1)
\end{aligned}
$$

Therefore, the two bit vectors B_1 and B_2 occupy $2m\log_2 3 + o(m)$ bits.

In addition, we store S_0, S_1 and S_2 using Lemma 2. The space cost of storing these three sequences is $2n + o(n) + 2(n-2) + o(n) + 2(n-3) + o(n) = 2m + o(m)$ bits. Thus the total space cost is $2m\log_2 6 + o(m)$ bits.

B_1 and B_2 can be used to compute the rank/select operations over S if we treat each type of (opening and closing) parentheses as the same character. For example, to compute the number of parentheses of the third type in $S[1..i]$, we can first compute the number of 0s in $S[1..i]$ (j denotes the result). Then the number of parentheses of the third type in $S[1..i]$ is $\texttt{rank}_{B_2}(0, j)$. Other rank/select operations can be supported similarly. On the other hand, S_0, S_1 and S_2 can be used to support operations on the parentheses of the same type. By representing all these data structures, the operations listed in Sect. 3.3 can be easily supported in constant time. As we store S_0 explicitly in our representation, we can trivially support the computation of an arbitrary word of S_0. □

The same approach can be directly applied to a sequence of $O(1)$ types of parentheses that may be unbalanced.

Lemma 12 *Consider a multiple parenthesis sequence M of n parenthesis of p types, where $p = O(1)$. M can be stored using $n\log(2p) + o(n)$ bits to support the operators listed in Sect. 3.3 in $O(1)$ time, as well as the computation of an arbitrary word, or $\Theta(\lg(n))$ bits of the balanced parenthesis sequence of the parentheses of a given type in M in $O(1)$ time.*

Proof Let n_i be the number of parentheses of type i in M. Let $l_i = \sum_{j=i}^{p} n_j$. Thus $l_1 = n$ and $l_p = n_p$. For $i = 1, 2, \ldots, p-1$, we construct a bit vector $B_i[1..l_i]$, where $B_i[k] = 1$ iff the kth parenthesis among the parentheses of types $i, i+1, \ldots, p$ in M is of type i. We store all the B_is using Part (b) of Lemma 1. Thus the space cost of all the B_is is $\sum_{i=1}^{p-1}\left[\lg\binom{l_i}{n_i} + o(l_i)\right] < \sum_{i=1}^{p-1}\left[\log_2\binom{l_i}{n_i} + 1 + o(n)\right] = \sum_{i=1}^{p-1}\log_2\binom{l_i}{n_i} + o(n)$. To analyze the above space cost, we again use the equality $\log_2 n! = n\log_2 n -$

$n \log_2 e + \frac{1}{2} \log_2 n + O(1)$ [15] (let $H_0^*(M)$ be the zeroth order entry of M when we replace each occurrence of the parentheses of the same type by one distinct character). We have:

$$
\begin{aligned}
&\sum_{i=1}^{p} \log_2 \binom{l_i}{n_i} \\
&= \log_2 \prod_{i=1}^{p} \binom{l_i}{n_i} \\
&= \log_2 \prod_{i=1}^{p} \frac{l_i!}{n_i!(l_i - n_i)!} \\
&= \log_2 \prod_{i=1}^{p} \frac{l_i!}{n_i! l_{i+1}!} \\
&= \log_2 \left(\frac{l_1!}{n_1! l_2!} \times \frac{l_2!}{n_2! l_3!} \times \cdots \times \frac{l_{p-1}!}{n_{p-1}! l_p!} \right) \\
&= \log_2 \frac{n!}{n_1! \times n_2! \times \cdots \times n_p!} \\
&= \log_2 n! - \sum_{i=1}^{p} \log_2 (n_i!) \\
&= n \log_2 n - n \log_2 e + \frac{1}{2} \log_2 n \\
&\quad - \sum_{i=1}^{p} \left(n_i \log_2 n_i - n_i \log_2 e + \frac{1}{2} \log_2 n_i \right) + O(1) \\
&= n \log_2 n - n \log_2 e + \frac{1}{2} \log_2 n \\
&\quad - \left[\sum_{i=1}^{p} (n_i \log_2 n_i) - n \log_2 e + \sum_{i=1}^{p} \frac{1}{2} \log_2 n_i \right] + O(1) \\
&= n \log_2 n - \sum_{i=1}^{p} (n_i \log_2 n_i) + O(\log_2 n) \\
&= \sum_{i=1}^{p} \left(n_i \log_2 \frac{n}{n_i} \right) + O(\log_2 n) \\
&= n H_0^*(M) + O(\log_2 n) \\
&\leq n \log_2 p + O(\log_2 n)
\end{aligned}
$$

Thus the space cost of all the B_is is $n \log_2 p + o(n)$ bits.

Let M_i be the subsequence of M that contains all the parentheses of type i. Note that M_i may be unbalanced. We use the approach of Chuang *et al.* [10] to encode all the M_is while supporting all the operations on balanced parentheses listed in Sect. 3.2 on them. More precisely, we insert one or more opening parentheses before the beginning of M_i and one or more closing parentheses after the end of M_i to get the shortest superstring, M_i', of M_i that is a balanced parenthesis sequence. Let n_i' denote the length of M_i'. Then we have $n_i' \leq 2n_i$. To encode M_i' while allowing the computation of any $O(\lg n)$-bit substring of $M_i'[j]$ in constant time, we need only store M_i and the numbers of opening and closing parentheses we insert before and after M_i respectively, which occupy $n_i + 2\lg n$ bits. We also build the auxiliary data structures for M_i' using Lemma 2. Thus it takes $n_i + 2\lg n + O(n_i' \lg\lg n_i' / \lg n_i')$ bits to encode M_i while supporting all the operations on balanced parentheses listed in Sect. 3.2 on M_i. Hence the space cost of all the M_is is $n + o(n)$ bits. Therefore, the total space cost of all the data structures is $n \log_2(2p) + o(n)$ bits.

As we can perform rank/select operations for each type of parentheses in M using B_is, and we can support all the operations on balanced parentheses listed in Sect. 3.2 on M_is, the algorithms used in the proof of Lemma 11 can be used to support the operators listed in Sect. 3.3 on M in $O(1)$ time. An arbitrary word of the parenthesis sequence of type i in M can be computed using M_i. □

The following theorem shows how to support the navigational operations on triangulations. While the space used here is a little more than that of Chiang *et al.* [9] (see Sect. 2), the explicit use of the three parenthesis sequences seems crucial to exploiting the realizers to support $\Pi_j(i)$ and $\Pi_j^{-1}(i)$ efficiently ($j \in \{1, 2\}$).

Theorem 1 *A planar triangulation $\mathcal{T}$ of n vertices and m edges can be represented using $2m \log_2 6 + o(m)$ bits to support operators* `adjacency`, `degree`, `select_neighbor_ccw`, `rank_neighbor_ccw` *as well as the $\Pi_j(i)$ and $\Pi_j^{-1}(i)$ ($j \in \{1, 2\}$) in $O(1)$ time.*

Proof We construct the string S for $\mathcal{T}$ as shown in this section, and store it using $2m \log_2 6 + o(m)$ bits by Lemma 11. Recall that S_0 is the balanced parenthesis encoding of $\overline{T_0}$, and that we can compute an arbitrary word of S_0 from S. Thus we can construct additional auxiliary structures using $o(n) = o(m)$ bits [9, 21–23] to support the navigational operations on $\overline{T_0}$. As each vertex is denoted by its rank in canonical ordering, vertex x corresponds to the xth opening parenthesis in S_0. We now show that these data structures are sufficient to support the navigational operations on $\mathcal{T}$.

To compute `adjacency`(x, y), recall that x and y are adjacent iff one is the parent of the other in one of the trees $\overline{T_0}$, T_1 and T_2. As S_0 encodes the balanced parenthesis sequence of $\overline{T_0}$, we can trivially check whether x (or y) is the parent of y (or x) using the `enclose` operator on S_0. To test adjacency in T_1, we recall that x is the parent of y iff the (only) outgoing edge of y, denoted by a $']'$, is an incoming edge of x, denoted by a $'['$. It then suffices to retrieve the first $']'$ after the yth $'('$ in S, given by `m_first`$(']')$, `m_select`$(y, '(')$, and compute the index, i, of its matching opening parenthesis, $'['$, in S. We then check whether the nearest succeeding closing parenthesis $')'$ of the $'['$ retrieved, located using `m_first`$(')', i)$, matches the xth

opening parenthesis $'('$ in S. If it does, then x is the parent of y in T_1. We use a similar approach to test the adjacency in T_2.

To compute `degree`(x), let d_0, d_1 and d_2 be the degrees (number of adjacent nodes including the parent) of x in the trees $\overline{T_0}$, T_1 and T_2 respectively, so that the sum of these three values is the answer. To compute d_0, we use S_0 and the algorithm to compute the degree of a node in an ordinal tree using its balanced parenthesis representation by Chiang *et al.* [9] (this is done using an operator called `wrapped`(i) on S_0, which returns the number of matching parenthesis pairs whose closest enclosing matching parenthesis pair has an opening parenthesis at position i). To compute $d_1 + d_2$, if x has children in $\overline{T_0}$, we first compute the indices, i_1 and i_2, of the xth and the $x+1$th $'('$ in S, and the indices, j_1 and j_2, of the $(n-x)$th and the $(n-x+1)$th $')'$ in S in constant time. By the third item of Property 1, we have the property $d_1 + d_2 = (i_2 - i_1 - 1) + (j_2 - j_1 - 1)$. The case in which x is a leaf in $\overline{T_0}$ can be handled similarly.

To support `select_neighbor_ccw` and `rank_neighbor_ccw`, we make use of the local condition of realizers in Definition 1. The local condition tells us that, given a vertex x, its neighbors, when listed in ccw order, form the following six types of vertices: x's parent in $\overline{T_0}$, x's children in T_2, x's parent in T_1, x's children in $\overline{T_0}$, x's parent in T_2, and x's children in T_1. The ith child of x in ccw order in $\overline{T_0}$ can be computed in constant time, and the number of siblings before a given child of x in ccw order can also be computed in constant time using the algorithms of Lu and Yeh [21]. The children of x in T_1 correspond to the parentheses $'['$ inserted before the parenthesis $')'$ corresponding to x when we construct S. In addition, by the construction of S, if u and v are both children of x, and u occurs before v in π_1, then u is also before v in ccw order among x's children in T_1. The children of x in T_2 have a similar property. Thus the operators supported on S allow us to perform rank/select on x's children in T_1 and T_2 in ccw order. As we can also compute the number of each type of neighbors of x in constant time, this allows us to support `select_neighbor_ccw` and `rank_neighbor_ccw` in $O(1)$ time.

To compute $\Pi_1(i)$, we first locate the position, j, of the ith occurrence of $'('$ in S, which is `m_select`$(i, '(')$. We then locate the position, k, of the first $']'$ after position j, which is `m_first`$(']', j)$. After that, we locate the matching parenthesis of $S[k]$ using `m_match`(k) (p denotes the result). $S[p]$ is the parenthesis $'['$ that corresponds to the edge between v_i and its parent in T_1, and by the construction algorithm of S, the rank of $S[p]$ is the answer, which is `m_rank`$(p, '[')$. The computation of Π_1^{-1} is exactly the inverse of the above process. Π_2 and Π_2^{-1} can be supported similarly. □

4.3 Vertex Labeled Planar Triangulations

We now consider a vertex labeled planar triangulation. Let n and m denote the numbers of its vertices and edges respectively, σ denote the number of labels, and t denote the total number of vertex-label pairs. As with binary relations [2, 3], we assume that each vertex is associated with at least one label.[4]

[4] As our approach reduces the support of operations on vertex labeled planar triangulations to the support of operations on binary relations, the same technique for binary relations [2, 3] can be used to generalize our results to the case in which each vertex is associated with zero or more labels.

In addition to unlabeled operators, we present a set of operators that allow efficient navigation in a vertex labeled planar triangulation (these are natural extensions to navigational operators on multi-labeled trees):

- `lab_degree`(α, x), the number of neighbors of vertex x that are labeled α;
- `lab_select_ccw`(α, x, y, r), the rth vertex labeled α among neighbors of vertex x after vertex y in ccw order, if y is a neighbor of x, and ∞ otherwise;
- `lab_rank_ccw`(α, x, y, z), the number of neighbors of vertex x labeled α between vertices y and z in ccw order if y and z are neighbors of x, and ∞ otherwise.

We define the interface of the ADT of vertex labeled planar triangulations through the operator `vertex_label`(v, r), which returns the rth label in lexicographic order associated with vertex v (i.e. the vth vertex in canonical ordering).

Recall that Lemma 11 encodes the string S constructed in Sect. 4.2 to support the computation of an arbitrary word of S_0, which is the balanced parenthesis sequence of the tree $\overline{T_0}$. In this section, we consider the `DFUDS` sequence [4] of $\overline{T_0}$, as the `DFUDS` order traversal visits the children of a node consecutively. We have the following lemma.

Lemma 13 *The string S can be stored in $(2\log_2 6 + \epsilon)m + o(m)$ bits, for any ϵ such that $0 < \epsilon < 1$, to support the operators listed in Sect. 3.3 in $O(1)$ time, as well as the computation of an arbitrary word, or $\Theta(\lg n)$ bits of the balanced parenthesis sequence, and of the* `DFUDS` *sequence of $\overline{T_0}$ in $O(1)$ time.*

Proof We construct the same data structures as in Lemma 11, except when we encode S_0 we use the `TC` representation of the tree $\overline{T_0}$ [16]. More precisely, we encode S_0 using $(2+\epsilon)n + o(n)$ bits, for any ϵ such that $0 < \epsilon < 1$, and this encoding supports the computation of an arbitrary word of the balanced parenthesis sequence, and the `DFUDS` sequence of $\overline{T_0}$ in constant time. As we can compute an arbitrary word of the original sequence of S_0 in constant time and all the other structures are the same as in Lemma 11, we can still support the operators listed in Sect. 3.3 in constant time. □

We now construct succinct indexes for vertex labeled planar triangulations.

Theorem 2 *Consider a multi-labeled planar triangulation $\mathcal{T}$ of n vertices, associated with σ labels in t pairs ($t \geq n$). Given the support of* `vertex_label` *in $f(n, \sigma, t)$ time on the vertices of $\mathcal{T}$, there is a succinct index using $t \cdot o(\lg \sigma) + O(t)$ bits which supports* `lab_degree`, `lab_select_ccw` *and* `lab_rank_ccw` *in $O((\lg\lg\lg\sigma)^2(f(n, \sigma, t) + \lg\lg\sigma))$ time.*

Proof The main idea is to combine our succinct representation of planar triangulations with three instances of the succinct indexes for related binary relations.

We represent the combinatorial structure of $\mathcal{T}$ using Theorem 1, in which we use Lemma 13 to store S. Thus we can construct the auxiliary data structures for the `DFUDS` representation of $\overline{T_0}$ [2–4, 19]. Observe that ranks of the vertices (for simplicity we consider only internal vertices) in three different orders, namely the `DFUDS` order of the nodes of $\overline{T_0}$, π_1 and π_2, form three binary relations, R_0, R_1 and R_2, with the labels associated with the corresponding vertices.

We adopt the same strategy used previously for multi-labeled trees [2, 3]. We can convert between the ranks of the vertices between π_0, π_1 and π_2 in constant time by Theorem 1. We can also convert between the preorder ranks of the nodes in $\overline{T_0}$ (note that they are in the order of π_0) and the DFUDS ranks of the nodes in $\overline{T_0}$ in constant time [2, 3]. Therefore, we can use the operator `vertex_label` to support the ADT of R_0, R_1 and R_2. Thus, for each of the three binary relations R_0, R_1 and R_2 we construct a succinct index of $t \cdot o(\lg \sigma)$ bits using Lemma 4.

To compute `lab_degree`(α, x), we first check whether x's parents in $\overline{T_0}$, T_1 and T_2 are labeled α. The DFUDS rank of x's parent in $\overline{T_0}$ can be computed in constant time. The rank in π_1 (or π_2) of x's parent in T_1 (or T_2) can also be computed in constant time, as shown in the proof of Theorem 1. Thus we can check whether x's parents in $\overline{T_0}$, T_1 and T_2 are labeled α by performing `label_access` operation on R_0, R_1 and R_2. We now need compute the numbers of x's children in $\overline{T_0}$, T_1 and T_2 that are associated with label α. By Lemma 9, x's children in $\overline{T_0}$, T_1 and T_2 are listed consecutively in DFUDS order of $\overline{T_0}$, π_1 and π_2, respectively. Compute the DFUDS rank, s, of x's first child in $\overline{T_0}$ in constant time. Then the DFUDS ranks of x's children in $\overline{T_0}$ are in the range $[s..s + d_0 - 1]$, where d_0 is the number of x's children within $\overline{T_0}$. Thus we can compute the number of x's children in $\overline{T_0}$ that are associated with label α by performing `label_rank` on R_0. To get the ranks in π_1 of x's children in T_1, we locate the first and last occurrences of parenthesis $'['$ inserted for the edges in T_1 whose parent node is x, and compute their ranks, f and l, among all the occurrences of parenthesis $'['$ in S. As the ranks in π_1 of x's children in T_1 are in the range $[f..l]$, we can compute the number of x's children in T_1 that are associated with label α by performing `label_rank` on R_1. The number of x's children in T_2 that are associated with label α can be computed similarly.

To support `lab_select_ccw` and `lab_rank_ccw`, by the local condition in Definition 1 and the algorithms in the above paragraph, it suffices to show that we can support the label-based rank/select of the children of a given node in ccw order in the three trees $\overline{T_0}$, T_1 and T_2, respectively. As we can compute the ranges of the DFUDS ranks in $\overline{T_0}$, the ranks in π_1 and the ranks in π_2 of x's children in $\overline{T_0}$, T_1 and T_2, respectively, these operations can be supported by performing `label_rank` and `label_select` operations on R_0, R_1 and R_2.

Finally, we observe that the space requirement of our representation is dominated by the cost of the succinct indexes for the binary relations, each using $t \cdot o(\lg \sigma) + O(t)$ bits. □

When σ is non-constant, the size of the succinct index constructed above becomes $t \cdot o(\lg \sigma)$ bits. To design a succinct representation of vertex labeled graphs using the above theorem, we have the following corollary:

Corollary 1 *A multi-labeled planar triangulation $\mathcal{T}$ of n vertices, associated with σ labels in t pairs ($t \geq n$) can be represented using $\lg \binom{n\sigma}{t} + t \cdot o(\lg \sigma) + O(t)$ bits to support* `vertex_label` *in $O(1)$ time, and* `lab_degree`, `lab_select_ccw` *and* `lab_rank_ccw` *in $O((\lg \lg \lg \sigma)^2 \lg \lg \sigma)$ time.*

Proof We use Lemma 5 to encode the binary relation between the vertices in canonical order and the set of labels in $\lg \binom{n\sigma}{t} + t \cdot o(\lg \sigma) + O(t)$ bits to support

`object_select` on it in constant time. Observe that the above operator directly supports `vertex_label` on $\mathcal{T}$. We then build the succinct indexes of $t \cdot o(\lg \sigma)$ bits for $\mathcal{T}$ using Theorem 2 and the corollary directly follows. □

As the information-theoretic lower bound of encoding the relation between the set of vertices and the set of labels in a vertex labeled planar triangulation is $\lg \binom{n\sigma}{t}$ bits, our result is close to the information-theoretic minimum of representing a vertex labeled planar triangulation.

4.4 Edge Labeled Planar Triangulations

In this section, we consider an edge labeled planar triangulation with n vertices and m edges. Let σ denote the number of labels, and t denote the total number of edge-label pairs. We adopt the assumption that each edge is associated with at least one label. We define the interface of the ADT of edge labeled planar triangulations through the operator `edge_label`(x, y, r), which returns the rth label associated to the edge between the vertices x and y in lexicographic order if they are adjacent, or 0 otherwise.

We consider the following operations:

- `lab_adjacency`(α, x, y), whether there is an edge labeled α between vertices x and y;
- `lab_degree_edge`(α, x), the number of edges incident with vertex x that are labeled α;
- `lab_select_edge_ccw`(α, x, y, r), the rth edge labeled α among edges incident with vertex x after edge (x, y) in ccw order, if y is a neighbor of x, and ∞ otherwise;
- `lab_rank_edge_ccw`(α, x, y, z), the number of edges incident with vertex x labeled α between edges (x, y) and (x, z) in ccw order if y and z are neighbors of x, and ∞ otherwise.

We construct the following succinct index for edge labeled planar triangulations.

Theorem 3 *Consider a multi-labeled planar triangulation $\mathcal{T}$ of n vertices and m edges, in which the edges are associated with σ labels in t pairs ($t \geq m$). Given the support of* `edge_label` *in $f(n, \sigma, t)$ time on the edges of $\mathcal{T}$, there is a succinct index using $t \cdot o(\lg \sigma) + O(t)$ bits which supports* `lab_adjacency` *in $O(\lg \lg \lg \sigma \cdot f(n, \sigma, t) + \lg \lg \sigma)$ time, and* `lab_degree_edge`, `lab_select_edge_ccw` *and* `lab_rank_edge_ccw` *in $O((\lg \lg \lg \sigma)^2 (f(n, \sigma, t) + \lg \lg \sigma))$ time.*

Proof We represent the combinatorial structure of $\mathcal{T}$ using Theorem 1, in which we use Lemma 13 to store S. We also construct the auxiliary data structures for the `DFUDS` representation of $\overline{T_0}$ [2–4, 19].

We number the edges in $\overline{T_0}$, T_1 and T_2 by the ranks of their child nodes in `DFUDS` order of $\overline{T_0}$, π_1 and π_2, respectively, and denote these three orders of edges by π'_0, π'_1 and π'_2, respectively. For example, in Fig. 2, the 8th edge in π'_0 is the edge between v_5 (i.e. the 8th node in `DFUDS` order of $\overline{T_0}$) and v_2. We observe that the ranks of the

edges in π'_0, π'_1 and π'_2 are from the sets $[n-1]$, $[n-2]$ and $[n-3]$, respectively. Thus the edges in π'_0, π'_1 and π'_2 and the label set $[\sigma]$ form three binary relations R'_0, R'_1 and R'_2, respectively. To support `object_select`(x, r) on R'_0, let y be the vertex whose `DFUDS` rank in $\overline{T_0}$ is x. We locate y's parent z, and `edge_label`(y, z, r) is the result. The support for `object_select` on R'_1 and R'_2 is similar. Therefore, we can use `edge_label` to support the ADT of R'_0, R'_1 and R'_2. For each of these three binary relations, we construct a succinct index of $t \cdot o(\lg\sigma) + O(t)$ bits using Lemma 4.

To compute `lab_adjacency`(α, x, y), we first use the algorithm in the proof of Theorem 1 to check whether x and y are adjacent, and if they are, which of the three trees ($\overline{T_0}$, T_1 and T_2) has the edge (x, y). If x is y's parent in $\overline{T_0}$, we compute y's `DFUDS` rank (i.e. the rank of edge (x, y) in π'_0), u, in $\overline{T_0}$, and `label_access`$_{R'_0}(u, \alpha)$ is the answer. The case in which x is y's child in $\overline{T_0}$, and the case in which the edge (x, y) is in T_1 or T_2 can be handled similarly. Thus, we can support `lab_adjacency` in $O(\lg\lg\lg\sigma \cdot f(n, \sigma, t) + \lg\lg\sigma)$ time.

To support the other three operations, we observe that the edges between a given vertex x and its children in $\overline{T_0}$, T_1 and T_2 have consecutive ranks in π'_0, π'_1 and π'_2, respectively. We also have x's children in $\overline{T_0}$ and T_1 are listed in ccw order in π'_0 and π'_1, respectively, and x's children in T_2 are listed in cw order in π_2. Thus we can use algorithms similar to those in Theorem 2 to support these operations.

Finally, we observe that the space requirement of our representation is dominated by the cost of the succinct indexes for the binary relations, each using $t \cdot o(\lg\sigma) + O(t)$ bits. □

To design a succinct representation of edge labeled graphs using the above theorem, we have the following corollary.

Corollary 2 *A multi-labeled planar triangulation $\mathcal{T}$ of n vertices and m edges, in which the edges are associated with σ labels in t pairs ($t \geq m$), can be represented using* $\lg\binom{m\sigma}{t} + t \cdot o(\lg\sigma) + O(t)$ *bits to support* `edge_label` *in* $O(1)$ *time,* `lab_adjacency` *in* $O(\lg\lg\sigma)$ *time, and* `lab_degree_edge`, `lab_select_edge_ccw` *and* `lab_rank_edge_ccw` *in* $O((\lg\lg\lg\sigma)^2 \lg\lg\sigma)$ *time.*

Proof We represent the combinatorial structure of $\mathcal{T}$ using Theorem 1, in which we use Lemma 13 to store S. Let m_1, m_2 and m_3 denote the number of edges in $\overline{T_0}$, T_1 and T_2, respectively. Let t_1, t_2 and t_3 denote the total numbers of edge-label pairs in $\overline{T_0}$, T_1 and T_2, respectively. We encode the three binary relations R'_0, R'_1 and R'_2 defined in Theorem 3 using Lemma 5. The space cost of encoding them in bits is $\sum_{i=0}^{2}\left[\lg\binom{m_i\sigma}{t_i} + t_i \cdot o(\lg\sigma) + O(t_i)\right] = \sum_{i=0}^{2}\left[\log_2\binom{m_i\sigma}{t_i} + t_i \cdot o(\lg\sigma) + O(t_i)\right] < \lg\binom{m\sigma}{et} + t \cdot o(\lg\sigma) + O(t)$, which dominates the overall space cost.

To support `edge_label`(x, y, r), we check which of the three trees, $\overline{T_0}$, T_1 and T_2, contains (x, y), and we compute the rank of this edge in π'_0, π'_1 or π'_3. We then perform `object_select` on R'_0, R'_1 or R'_2 to compute the result. The other operations can be supported using Theorem 3. □

4.5 Extensions to Planar Graphs

We now extend the techniques of Sects. 4.2, 4.3 and 4.4 to general planar graphs. As any planar graph can be embedded in the plane (i.e. drawn in the plane without edge intersections), it suffices to represent *plane graphs* which are planar graphs already embedded in the plane.

Consider a plane graph G of n vertices and m edges. To use our results on planar triangulations, we construct a planar triangulation $\mathcal{T}$ for G using the following approach. We first surround G with a large triangle such that all the vertices and edges of G are in the interior of this triangle. We add the three vertices of this triangle and the three edges between them into G, and denote the resulting graph G'. Finally, we triangulate each interior face of G' that is a polygon with more than three vertices. The resulting graph is the planar triangulation $\mathcal{T}$.

Let n' and m' be the number of vertices and edges of $\mathcal{T}$, respectively. Then we have $n' = n + 3$ and $m' = 3n + 3$. We denote the three vertices on the exterior face of $\mathcal{T}$ by v_0, v_1 and v_{n+2}. We denote the vertices of G by their ranks in the canonical ordering of $\mathcal{T}$. Thus the vertices of G are $v_2, v_3, \ldots, v_{n+1}$. The three orders π_0, π_1 and π_2 on the vertices of G are simply given by these three orders on the vertices of $\mathcal{T}$. Recall that we use (T_0, T_1, T_2) to denote the realizer of $\mathcal{T}$, and $\overline{T_0}$ to denote its canonical spanning tree.

We first extend Theorem 1 to represent unlabeled plane graphs.

Theorem 4 *A plane graph G of n vertices and m edges can be represented using $3n(2\log_2 3 + 3 + \epsilon) + o(n)$ bits to support operators* `adjacency`, `degree`, `select_neighbor_ccw`, `rank_neighbor_ccw` *as well as $\Pi_j(i)$ and $\Pi_j^{-1}(i)$ ($j \in \{1, 2\}$) in $O(1)$ time.*

Proof We construct the planar triangulation $\mathcal{T}$ for G using the above approach. We then represent $\mathcal{T}$ using Theorem 1, in which we use Lemma 13 to encode the string S constructed to encode $\mathcal{T}$. Thus $\mathcal{T}$ is encoded in $m'(2\log_2 6 + \epsilon) + o(m') = 3n(2\log_2 6 + \epsilon) + o(n)$ bits. In addition, we construct the following three bit vectors to indicate which edge in $\mathcal{T}$ is present in G:

- A bit vector $B_0[1..n+2]$, where $B_0[i] = 1$ iff the edge between the ith vertex in `DFUDS` order of $\overline{T_0}$ and its parent in $\overline{T_0}$ is present in G;
- A bit vector $B_1[1..n+1]$, where $B_1[i] = 1$ iff the edge between the ith vertex in π_1 and its parent in T_1 is present in G;
- A bit vector $B_2[1..n]$, where $B_2[i] = 1$ iff the edge between the ith vertex in π_2 and its parent in T_2 is present in G.

We encode these three bit vectors in $3n + o(n)$ bits using Part (a) of Lemma 1. Thus the total space cost is $3n(2\log_2 6 + \epsilon) + 3n + o(n) = 3n(2\log_2 3 + 3 + \epsilon) + o(n)$ bits.

To compute `adjacency`(x, y), we first check whether x and y are adjacent in $\mathcal{T}$. If they are not, we return false. If they are, the algorithm in the proof of Theorem 1 also tells us which of the three trees, $\overline{T_0}$, T_1 and T_2, has the edge (x, y) of $\mathcal{T}$. If x is y's parent in $\overline{T_0}$, we compute y's `DFUDS` rank, j, in $\overline{T_0}$. If $B_0[j] = 1$, then the edge (x, y) is in G, so we return true. We return false otherwise. The case in which y is x's

parent in $\overline{T_0}$, and the case in which the edge (x, y) of $\mathcal{T}$ is in T_1 or T_2 can be handled similarly.

To compute `degree`(x), we observe that the algorithm in the above paragraph can be used to check whether x and its parents in $\overline{T_0}$, T_1 and T_2 are adjacent in G. Thus it suffices to compute the number of x's children in $\overline{T_0}$, T_1 and T_2 that are adjacent to x in G. To count the number, u, of x's children in $\overline{T_0}$ that are adjacent to x in G, we compute the `DFUDS` ranks, p and q, of the first and the last child of x in $\overline{T_0}$. Then u is equal to the number of 1s in $B_0[p..q]$, which can be computed in constant time by performing `rank` on B_0. The number of x's children in T_1 or T_2 that are adjacent to x in G can be computed similarly.

To use the algorithms in the proof of Theorem 1 to support `select_neighbor_ccw` and `rank_neighbor_ccw`, it suffices to support these two operations: given a vertex x, select its ith child in $\overline{T_0}$ (T_1 or T_2) that is adjacent to it in G; given a vertex x and a child, y, of it in $\overline{T_0}$ (T_1 or T_2) that is adjacent to x in G, compute the number of y's left siblings that are adjacent to x in G. To support these two operations, we first compute the `DFUDS` ranks in $\overline{T_0}$ (ranks in π_1 or π_2) of the first and last children of x in $\overline{T_0}$ (T_1 or T_2). From these, we can locate the substring of B_0 (B_1 or B_2) corresponding to the children of x in $\overline{T_0}$ (T_1 or T_2), and perform rank/select operations on it to support these two operations in constant time.

Finally, we observe that the algorithms in the proof of Theorem 1 to support Π_j and Π_j^{-1} on planar triangulations can be used here directly. □

We construct the following succinct indexes for vertex labeled plane graphs.

Theorem 5 *Consider a multi-labeled plane graph G of n vertices, associated with σ labels in t pairs ($t \geq n$). Given the support of* `vertex_label` *in $f(n, \sigma, t)$ time on the vertices of G, there is a succinct index using $t \cdot o(\lg \sigma) + O(t)$ bits which supports* `lab_degree`, `lab_select_ccw` *and* `lab_rank_ccw` *in $O((\lg\lg\lg\sigma)^2(f(n, \sigma, t) + \lg\lg\sigma))$ time.*

Proof We represent the combinatorial structure of G using Theorem 4. The vertices of G in canonical order and the set of labels $[\sigma]$ form a binary relation L. As `vertex_label` directly supports `object_select` on L, we construct a succinct index of $t \cdot o(\lg \sigma) + O(t)$ bits using Lemma 4 for L.

In addition, we construct three binary relations, L_0, L_1 and L_2, between the ranks of the vertices of G in three different orders and the set of labels. In L_0, the ith object corresponds to the ith vertex in `DFUDS` order of $\overline{T_0}$. If this vertex and its parent in $\overline{T_0}$ are adjacent in G, we associate its labels with the ith object. Otherwise, we do not associate any label with this object. As we can perform constant time conversions between the canonical order of a vertex and its `DFUDS` rank in $\overline{T_0}$, and we can also check whether a node and its parent in $\overline{T_0}$ are adjacent in G, we can use `vertex_label` to support `object_select` on L_0. We construct L_1 and L_2 using the same approach, except that the ith object in L_1 and L_2 corresponds to the ith vertex in π_1 and π_2, respectively. We can also use `vertex_label` to support `object_select` on them. We construct a succinct index of $t \cdot o(\lg \sigma) + O(t)$ bits using Lemma 4 for each of these three binary relations. Note that although Lemma 4

assumes that each object is associated with at least one label, the techniques created to prove it still applies to the more general case in which an object is associated with zero or more labels (see Sect. 3.4). Furthermore, the result is the same asymptotically if $t > n$, which is true here.

As we can perform conversions between the DFUDS rank of $\overline{T_0}$, π_0, π_1 and π_2, we can perform `label_access` on L to check whether the parent of a given vertex in $\overline{T_0}$, T_1 or T_2 is associated with a given label (if this vertex and the parent are adjacent in G). We also observe that if a vertex and one of its children in $\overline{T_0}$, T_1 or T_2 are not adjacent in G, then the object in L_0, L_1 or L_2 that corresponds to the child is not associated with any label. Thus we can use the algorithms in the proof of Theorem 2 to support `lab_degree`, `lab_select_ccw` and `lab_rank_ccw`, and this theorem follows. □

To design a succinct representation for a vertex labeled plane graph based on the above theorem, we can use the approach in the proof of Corollary 1, and the following corollary is immediate.

Corollary 3 *A multi-labeled plane graph G of n vertices, associated with σ labels in t pairs ($t \geq n$) can be represented using $\lg \binom{n\sigma}{t} + t \cdot o(\lg \sigma) + O(t)$ bits to support* `vertex_label` *in $O(1)$ time, and* `lab_degree`, `lab_select_ccw` *and* `lab_rank_ccw` *in $O((\lg\lg\lg\sigma)^2 \lg\lg\sigma)$ time.*

We now design succinct indexes for edge labeled plane graphs.

Theorem 6 *Consider a multi-labeled plane graph G of n vertices and m edges, in which the edges are associated with σ labels in t pairs ($t \geq m$). Given the support of* `edge_label` *in $f(n, \sigma, t)$ time on the edges of $\mathcal{T}$, there is a succinct index using $t \cdot o(\lg\sigma) + O(n+t)$ bits which supports* `lab_adjacency` *in $O(\lg\lg\lg\sigma f(n, \sigma, t) + \lg\lg\sigma)$ time, and* `lab_degree_edge`, `lab_select_edge_ccw` *and* `lab_rank_edge_ccw` *in $O((\lg\lg\lg\sigma)^2(f(n, \sigma, t) + \lg\lg\sigma))$ time.*

Proof We add labels to the edges of the planar triangulation $\mathcal{T}$ constructed in this section for G as follows. For each edge of $\mathcal{T}$ that is in G, we label it with its labels in G. For each edge of $\mathcal{T}$ that is not in G, we do not associate it with any label. We also construct B_0, B_1 and B_2 as in the proof of Theorem 4. As we can check whether an edge of $\mathcal{T}$ is in G in constant time, we can support `edge_label` on $\mathcal{T}$ in $f(n, \sigma, t)$ time using B_0, B_1 and B_2. We use Theorem 3 to construct a succinct index for the edge-labeled version of $\mathcal{T}$.

To analyze the space cost, we observe that to encode the succinct indexes for the three binary relations in the proof of Theorem 3, we need $t \cdot o(\lg\sigma) + O(t + m')$ bits in total (see the last paragraph in Sect. 3.4). It requires $O(n)$ bits to encode the combinatorial structure of $\mathcal{T}$. Each of the three bit vectors B_0, B_1 and B_2 occupies $n + o(n)$ bits. Thus the overall space cost is $t \cdot o(\lg\sigma) + O(t+n)$ bits.

Observe that although we add more edges when constructing $\mathcal{T}$, none of them is associated with any labels. Therefore, the operations `lab_adjacency`, `lab_degree_edge`, `lab_select_edge_ccw` and `lab_rank_edge_ccw` can be used directly to support the same operations on G, and the theorem follows. □

To design a succinct representation for an edge labeled plane graph based on the above theorem, we have the following corollary.

Corollary 4 *A multi-labeled plane graph G of n vertices and m edges, in which the edges are associated with σ labels in t pairs ($t \geq m$), can be represented using $\lg \binom{m\sigma}{t} + t \cdot o(\lg \sigma) + O(t+n)$ bits to support* `edge_label` *in $O(1)$ time,* `lab_adjacency` *in $O(\lg \lg \sigma)$ time, and* `lab_degree_edge`, `lab_select_edge_ccw` *and* `lab_rank_edge_ccw` *in $O((\lg \lg \lg \sigma)^2 \lg \lg \sigma)$ time.*

Proof We encode the combinatorial structure of the planar triangulation $\mathcal{T}$ using Theorem 1. We construct an edge-labeled version of $\mathcal{T}$ as in the proof of Theorem 6. Compute the realizer (T_0, T_1, T_2) of $\mathcal{T}$. Let m'_1, m'_2 and m'_3 denote the numbers of edges of $\mathcal{T}$ in $\overline{T_0}$, T_1 and T_2, respectively. Let m_1, m_2 and m_3 denote the numbers of edges of G in $\overline{T_0}$, T_1 and T_2, respectively. Let t_1, t_2 and t_3 denote the total numbers of edge-label pairs in $\overline{T_0}$, T_1 and T_2, respectively. We use the notion of the three orders, π'_0, π'_1 and π'_2 defined on the edges of $\mathcal{T}$ as in the proof of Theorem 6. We construct the three bit vectors B_0, B_1 and B_2 as in the proof of Theorem 4.

Consider the edges of G that are in $\overline{T_0}$. Observe that each of them corresponds to a 1 in B_0. These edges in the order of π'_0 and the set of labels form a binary relation and we use E'_0 to denote it. We use the approach of Barbay *et al.* [2, Proof of Theorem 7] to encode E'_0 in $O(m_1 + t_1) + \lg \binom{m_1\sigma}{t_1} + O(\lg \lg(n\sigma))$ bits to support `object_select` on it in constant time. Similarly, we define two binary relations E'_1 and E'_2 between the edges of G in T_1 and T_2 in the orders of π'_1 and π'_2, respectively, and the set of labels. We use the same approach to encode them. Thus the total space used to encode these three binary relations is $\sum_{i=0}^{2} \big(O(m_i + t_i) + \lg \binom{m_i\sigma}{t_i} + O(\lg \lg(n\sigma))\big) \leq O(m+t) + \lg \binom{m\sigma}{t} + O(\lg \lg(n\sigma))$ bits.

To use Theorem 6 to prove this corollary, it suffices to support `edge_label`(x, y, r) on G. As the case in which x and y are not adjacent in G is trivial, we consider only the case in which they are adjacent. We first consider the case in which the edge (x, y) is in $\overline{T_0}$. Assume, without the loss of generality, that x is y's parent in $\overline{T_0}$. Let j be y's `DFUDS` rank in $\overline{T_0}$. In this case, the edge (x, y) is numbered j in π'_0, which corresponds to the $(k = \mathtt{rank}_{B_0}(1, j))$th edge in E'_0. Thus $\mathtt{object\_select}_{E'_0}(k, r)$ is the result. The case in which the edge (x, y) is in T_1 or T_2 can be handled similarly.

The overall space is $t \cdot o(\lg \sigma) + O(t+n) + O(m+t) + \lg \binom{m\sigma}{t} + O(\lg \lg(n\sigma))$, which is $\lg \binom{m\sigma}{t} + t \cdot o(\lg \sigma) + O(t+n)$ bits as $t \geq m$. □

5 *k*-Page Graphs

5.1 Multiple Parentheses

To present our result on multiple parentheses, we first consider the following operation on strings: $\mathtt{string\_rank}'_S(\alpha, i)$, which returns the number of characters α in $S[1..i]$ if $S[i] = \alpha$ (the result is undefined otherwise). This is a less powerful version of `string_rank`. We have the following lemma.

Lemma 14 *A string S of length n over alphabet $[\sigma]$ can be represented using $n(H_0(S) + o(\lg\sigma) + O(1))$ bits to support* `string_access` *and* `string_rank` *in $O(\lg\lg\sigma)$ time, and* `string_rank'` *and* `string_select` *in $O(1)$ time. Given a character $\alpha \in [\sigma]$, this representation also supports the computation of the number of characters in S that are lexicographically smaller than α in $O(1)$ time.*

Alternatively, S can be represented using $n(H_0(S) + \epsilon\lg\sigma + o(\lg\sigma) + O(1))$ bits for any constant ϵ such that $0 < \epsilon < 1$ to support `string_access` *in $O(1)$ time, while providing the same support for all the other operations above.*

Proof To prove the result in the first paragraph of this lemma, we use the approach of Barbay *et al.* [2, Lemma 12] to encode the string S in $n(H_0(S) + o(\lg\sigma) + O(1))$ bits. This encoding supports `string_access` and `string_rank` in $O(\lg\lg\sigma)$ time, and `string_select` in $O(1)$ time. Thus we need only show how to support $\texttt{string\_rank}'(\alpha, i)$, and how to compute the number of characters of S that are lexicographically smaller than a given character α. In the approach of [2], S is conceptually treated as an $n \times \sigma$ table E, in which $E[\alpha][x] = 1$ iff $S[x] = \alpha$. The auxiliary structures constructed support the computation of $\texttt{rank}(1, i)$ on an arbitrary row, $E[\alpha]$, of E in constant time, if $E[\alpha][i] = 1$. This can be directly used to support $\texttt{string\_rank}'(\alpha, i)$ in constant time. The number of characters in S that are lexicographically smaller than α is equal to the number, p, of 1s in the first $\alpha - 1$ rows of E. Each row of E is further divided into *blocks* of length σ as noted in [2], and the *cardinality* of each block is defined as the number of 1s in it. An auxiliary structure is used to encode the cardinality of each block in row major order, and we can perform rank/select operations on this auxiliary structure to compute p in constant time.

To prove the second claim of this lemma, observe that the result of Barbay *et al.* [2] used in the previous paragraph makes use of a succinct index they designed for strings. In this index, the string S is divided into substrings called *chunks* of length σ, and a permutation π is defined for each chunk. The support for `string_access` is then reduced to the access to π^{-1}. An auxiliary structure, P, of $O(\sigma\lg\sigma/\lg\lg\sigma)$ bits is constructed using the approach of Munro *et al.* [24] for each chunk, so that any element of π and π^{-1} can be retrieved in $O(1)$ time and $O(\lg\lg\sigma)$ time respectively, when `string_select` is supported in constant time. Thus with this structure, `string_access` can be performed in $O(\lg\lg\sigma)$ time. If we increase the size of P to $\epsilon\sigma\lg\sigma$ bits, then each element of π and π^{-1} can be computed in constant time [24], so that `string_access` can be supported in $O(1)$ time. The total space cost of such auxiliary data structures for all the chunks then becomes $\epsilon n\lg\sigma$ bits, which accounts for the increase in the overall space cost of our representation of S. □

We now consider the succinct representations of multiple parenthesis sequences of p types of parentheses, where p is not a constant. We consider the following operation on a multiple parenthesis sequence $S[1..2n]$ in addition to those defined in Sect. 3.3:

- $\texttt{m\_access}(i)$, the parenthesis at position i;
- $\texttt{m\_rank}'_S(i)$, the rank of the parenthesis at position i among parentheses of the same type in S.

We have the following theorem.

Theorem 7 *A multiple parenthesis sequence of* $2n$ *parentheses of* p *types, in which the parentheses of any given type are balanced, can be represented using* $2n \lg p + n \cdot o(\lg p) + O(n)$ *bits to support* `m_access`, `m_rank′` *and* `m_match` *in* $O(\lg \lg p)$ *time, and* `m_select` *in* $O(1)$ *time. Alternatively,* $(2+\epsilon)n \lg p + n \cdot o(\lg p) + O(n)$ *bits are sufficient to support these operations in* $O(1)$ *time, for any constant* ϵ *such that* $0 < \epsilon < 1$.

Proof We store the sequence as a string P over alphabet $\{'(_1', ')_1', '(_2', ')_2', \ldots, '(_p', ')_p'\}$ using the result in the first paragraph of Lemma 14. P occupies at most $2n(\lg(2p) + o(\lg p) + O(1)) = 2n(\lg p + o(\lg p) + O(1))$ bits.

For each integer i such that $1 \le i \le p$, we construct a balanced parenthesis sequence B_i, where $B_i[j]$ is an opening parenthesis iff the jth parenthesis of type i in P is open. We denote the number of parentheses of type i by n_i. Then the length of B_i is n_i. To store all these sequences, we concatenate them to get a balanced parenthesis sequence B, and we store B in $2n + o(n)$ bits using Lemma 2. In order to locate B_i in B, it suffices to compute the numbers of characters in P that are lexicographically smaller than $'(_i'$ and $'(_{i+1}'$, which is supported in constant time by Lemma 14. This allows us to perform `find_open` and `find_close` operations on B_i in constant time.

The operation `m_access` can be supported by calling `string_access` on P once, so it can be supported in $O(\lg \lg p)$ time. To support $\texttt{m\_rank}'(i)$, we first compute the parenthesis, α, at position i using `m_access` in $O(\lg \lg p)$ time. Then $\texttt{m\_rank}'(i) = \texttt{string\_rank}'(\alpha, i)$. We also have $\texttt{m\_select}(\alpha, i) = \texttt{string\_select}(\alpha, i)$. Finally, to support $\texttt{m\_match}(i)$, we first find out which parenthesis is at position i using `m_access`. Assume, without loss of generality, it is an opening parenthesis, and let $'(_j'$ be this parenthesis. Then the index of the entry in B_j that corresponds to this parenthesis is $q = \texttt{select\_open}_{B_j}(\texttt{m\_rank}'('(_j', i))$, and we have $\texttt{m\_match}(i) = \texttt{m\_select}(')_j', \texttt{find\_close}_{B_j}(q))$.

To support all these operations in constant time, it suffices to support `string_access` on P in constant time. This can be achieved by using the result in the second paragraph of Lemma 14. The total space is thus increased by $\epsilon n \lg p$ bits. □

5.2 k-Page Graphs for Large k

On unlabeled k-page graphs, we consider the operators `adjacency` and `degree` defined in Sect. 4.2, and the operator $\texttt{neighbors}(x)$, returning the neighbors of x.

As mentioned in Sect. 2, previous results on representing k-page graphs [14, 22] succinctly support `adjacency` in $O(k)$ time. The lower-order term in the space cost of the result of Gavoille and Hanusse [14] is $o(km)$, which is dominant when k is large. Thus previous results mainly deal with the case in which k is small. We consider large k.

In this section, we denote each vertex of a k-page graph by its rank along the spine of the book (i.e. vertex x is the xth vertex along the spine). We define the *span* of an edge between vertices x and y to be $|y - x|$. An edge between vertices x and y, where $x < y$, is a *right edge* of x and a *left edge* of y. We show the following result.

Theorem 8 *A k-page graph G of n vertices and m edges can be represented using $n + 2m \lg k + m \cdot o(\lg k) + O(m)$ bits to support* `adjacency` *in $O(\lg k \lg\lg k)$ time,* `degree` *in $O(1)$ time, and* `neighbors`(x) *in $O(d(x) \lg\lg k)$ time where $d(x)$ is the degree of x. Alternatively, it can be represented in $n + (2+\epsilon)m \lg k + m \cdot o(\lg k) + O(m)$ bits to support* `adjacency` *in $O(\lg k)$ time,* `degree` *in $O(1)$ time, and* `neighbors`(x) *in $O(d(x))$ time, for any constant ϵ such that $0 < \epsilon < 1$.*

Proof We construct a bit vector B of $n+m$ bits to encode the degree of each vertex in unary as in [18], in which vertex x corresponds to the xth 1 followed by $d(x)$ 0s. We encode B in $n + m + o(n+m)$ bits using part (a) of Lemma 1 to support rank/select operations. We construct a multiple parenthesis sequence S of $2m$ parentheses of k types as follows. For each vertex $x \in \{1, 2, \ldots, n\}$, we write down the following four groups of parentheses as a substring of S in the order specified below:

1. For each page $i \in \{1, 2, \ldots, k\}$, if there are j left edges of x on this page where $j > 0$, we write down $j - 1$ copies of the symbol $')_i'$.
2. Assume that the left edges of x are on pages $p_1, p_2, \ldots, p_l$. We sort the sequence $')_{p_1}', ')_{p_2}', \ldots, ')_{p_l}'$ by the maximum span of the left edges of x on these pages and write down the sorted sequence, i.e. in the sorted sequence, $')_{p_u}'$ appears before $')_{p_v}'$ if the maximum span of the left edges of x on page p_u is less than the maximum span of the left edges of x on page p_v.
3. Similarly, we assume that the right edges of x are on pages $q_1, q_2, \ldots, q_r$. We sort the sequence $'(_{q_1}', '(_{q_2}', \ldots, '(_{p_l}'$ by the maximum span of the right edges of x on these pages and write down the sorted sequence in descending order.
4. For each page $i \in \{1, 2, \ldots, k\}$, if there are j' right edges of x on this page where $j' > 0$, we write down $j' - 1$ copies of $'(_i'$.

Although the sequence, S, appears similar to the sequence in Theorem 2 of [14], it differs in the order we store the parentheses corresponding to the edges of a given vertex. It also has $2m$ parentheses of k types, and we encode it using Theorem 7 in $2m \lg k + m \cdot o(\lg k) + O(m)$ bits. Finally we construct a bit vector B' of $2m$ bits in which $B'[i] = 1$ iff $S[i]$ is a closing parenthesis, and encoding it in $2m + o(m)$ bits using part (a) of Lemma 1 to support rank/select operations. Thus the total space cost is $n + 2m \lg k + m \cdot o(\lg k) + O(m)$ bits.

With the above definitions and structures, the algorithm [14] that checks whether there is an edge between vertices x and y on page p can be described as follows (assume, without loss of generality, that $x < y$). Let w be the index of the parenthesis in S that corresponds to the right edge of x with the largest span on page p. Observe that this position corresponds to the first occurrence of the character $(_p$ in S after position $\texttt{rank}_B(0, \texttt{select}_B(1, x))$. We assume that w is given and design the following algorithm, which is used in the next paragraph to support `adjacency`. We retrieve the index, t, of the closing parenthesis that matches $S[w]$ in $O(\lg\lg k)$ time, and if it corresponds to a left edge of y (this is true iff $\texttt{rank}_B(1, t) = y$), then there is an edge between x and y. Similarly, we retrieve the parenthesis in S that corresponds to the left edge of y with the largest span on page p (again, we assume that the index of the occurrence of the parenthesis corresponding to it is given), and if its matching opening parenthesis corresponds to a right edge of x, then x and y are adjacent. If the

above process cannot find an edge between x and y, then x and y are not adjacent on page p. All these steps take $O(\lg \lg k)$ time.

To compute $\texttt{adjacency}(x, y)$ (assume, without loss of generality, that $x < y$), we first observe that by Step 2 of the construction algorithm for S, the opening parentheses that correspond to the right edges of x with the largest spans among the right edges of x on the same pages form a substring of S. We can compute the starting position of this substring using B and B' in constant time. Because these parentheses are sorted by the spans of the edges they correspond to, we can perform a doubling search to check whether one of these edges connects x and y. In each step of the doubling search, we perform the algorithm in the previous paragraph in $O(\lg \lg k)$ time. There are at most k such parentheses, so we perform the algorithm $O(\lg k)$ times. Similarly, we perform doubling search on the left edges of y with the largest spans among the left edges of y on the same pages. Thus we can test the adjacency between two vertices in $O(\lg k \lg \lg k)$ time.

The degree of any vertex can be easily computed in constant time using B. We can also use the algorithms presented in previous work [14] to compute $\texttt{neighbors}(x)$. More precisely, for each opening or closing parenthesis corresponding to an edge incident with x, we find its matching parenthesis to locate the other vertex that it is incident with. This takes $O(d(x) \lg \lg k)$ time on our data structures.

Finally, to improve the time efficiency, we can store S using $(2+\epsilon)m \lg k + m \cdot o(\lg k) + O(m)$ bits using Theorem 7 to achieve the other tradeoff. □

5.3 Edge Labeled k-Page Graphs

On edge labeled k-page graphs, we consider `lab_adjacency` and `lab_degree_edge` defined in Sect. 4.4, as well as the following operation: $\texttt{lab\_edges}(\alpha, x)$, the edges incident with vertex x that are labeled α. We define the interface of the ADT of edge labeled k-page graphs through the operator `edge_label`, as defined in Sect. 4.4.

We first design a succinct index for an edge labeled graph with one page, i.e. an outerplanar graph.

Lemma 15 *Consider a multi-labeled outerplanar graph G of n vertices and m edges, in which the edges are associated with σ labels in t pairs $(t \geq m)$. Given the support of* `edge_label` *in $f(n, \sigma, t)$ time on the edges of G, there is a succinct index using $t \cdot o(\lg \sigma) + O(t) + n + o(n)$ bits which supports* `lab_adjacency` *in $O(\lg \lg \lg \sigma f(n, \sigma, t) + \lg \lg \sigma)$ time,* `lab_degree_edge` *in $O((\lg \lg \lg \sigma)^2 (f(n, \sigma, t) + \lg \lg \sigma))$ time, and* $\texttt{lab\_edges}(\alpha, x)$ *in $O(d(\lg \lg \lg \sigma)^2 \times (f(n, \sigma, t) + \lg \lg \sigma))$ time, where $d = \texttt{lab\_degree\_edge}(\alpha, x)$.*

Proof We construct a bit vector B of $n + m$ bits to encode the degree of each vertex in unary as in the proof of Theorem 8, and use part (a) of Lemma 1 to encode it. We construct a balanced parenthesis sequence P as follows. List the vertices from left to right along the spine, and each vertex is represented by zero or more closing parentheses followed by zero or more opening parentheses, where the numbers of closing and opening parentheses are equal to the numbers of its left and right edges respectively. The edges ordered by the positions of the corresponding opening parentheses

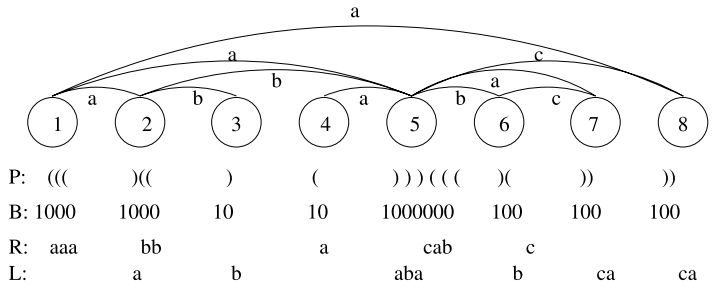

Fig. 5 An example of the succinct representation of a labeled graph with one page. For simplicity, each edge is associated with exactly one label in this example

and the set of labels form a binary relation R. Similarly, the edges ordered by the positions of the corresponding closing parentheses and the set of labels form a binary relation L. Figure 5 gives an example.

To compute $\texttt{object\_select}_R(x, r)$, we first find the two vertices y and z ($y < z$) incident with the edge corresponding to the xth opening parenthesis in P. As this parenthesis corresponds to the ith 0 in B, where $i = \texttt{select}_P('(', x)$, we have $y = \texttt{rank}_B(1, i)$. We find the closing parenthesis that matches this opening parenthesis in P using `find_close`, and z can be computed similarly. As $\texttt{object\_select}_R(x, r) = \texttt{edge\_label}(y, z, r)$, we can support `object_select` on R in $f(n, \sigma, t)$ time. The support for `object_select` on L is similar. We then build a succinct index of $t \cdot o(\lg \sigma) + O(t)$ bits for either of L and R using Lemma 4. These data structures occupy $t \cdot o(\lg \sigma) + O(t) + n + o(n)$ bits in total as $t \geq m$.

To compute $\texttt{lab\_adjacency}(\alpha, x, y)$, we first use the algorithm of Jacobson [18] to check whether x and y are adjacent. If they are, we retrieve the position of the opening parenthesis in P that corresponds to the edge between x and y, compute its rank, v, among opening parenthesis, and we return the result of $\texttt{label\_access}(v, \alpha)$ on R. This takes $O(\lg \lg \lg \sigma f(n, \sigma, t) + \lg \lg \sigma)$ time.

To compute $\texttt{lab\_degree\_edge}(\alpha, x)$, we need compute the number, l, of left edges of x labeled α, and the number, r, of right edges of x labeled α. To compute l, we first compute the positions l_1 and l_2 such that each parenthesis in the substring $P[l_1..l_2]$ is a closing parenthesis that corresponds to a left edge of x, using rank/select operations on B and P in constant time. We then use `label_rank` and `label_select` on L to compute the number of objects associated with α between and including objects l_1 and l_2 in $O((\lg \lg \lg \sigma)^2 (f(n, \sigma, t) + \lg \lg \sigma))$ time. Similarly we can compute r by performing rank/select operations on B, P and R, and the sum of l and r yields the result. To further list all the edges of x that are labeled α, we perform `label_rank` and `label_select` on L and R to retrieve the positions of the corresponding parentheses in P, and perform rank and select operations on B to retrieve the vertices incident with these edges. □

We now use the above lemma to design a succinct representation of edge labeled outerplanar graph.

Lemma 16 *An outerplanar graph of n vertices and m edges in which the edges are associated with σ labels in t pairs ($t \geq m$) can be represented using $n + o(n) + \lg \binom{m\sigma}{t} + t \cdot o(\lg \sigma) + O(t)$ bits to support*:

- `edge_label` *in* $O(1)$ *time*;
- `lab_adjacency` *in* $O(\lg \lg \sigma)$ *time*;
- `lab_degree_edge` *in* $O((\lg \lg \lg \sigma)^2 \lg \lg \sigma)$ *time*;
- `lab_edges`(α, x) *in* $O(d(\lg \lg \lg \sigma)^2 \lg \lg \sigma)$ *time*, *where* $d =$ `lab_degree_edge`(α, x).

Proof We construct B and P as in the proof of Lemma 15. We use Lemma 5 to represent the binary relation R defined in the proof of Lemma 15. This costs $\lg \binom{m\sigma}{t} + t \cdot o(\lg \sigma) + O(t)$ bits. Given two adjacent vertices x and y, we can locate the opening parenthesis corresponding to the edge between x and y using P and B in constant time. Thus we can use `object_select` on R to directly support `edge_label`, which in turn supports `object_select` on L. Hence we can construct a succinct index of $t \cdot \lg \sigma$ bits for the binary relation L defined in the proof of Lemma 15, and this lemma immediately follows. □

To represent an edge labeled k-page graph, we can use Lemma 16 to represent each page and combine all the pages represented in this way to support operations. Alternatively, we can use Theorem 8 and an approach similar to Lemma 16 to achieve a different tradeoff to improve the time efficiency for large k. As we consider general k, the auxiliary data structures may occupy more space than the labels themselves. Thus we choose to directly show our succinct representations instead of presenting a succinct index first. Note that for sufficiently small k, this approach can still be used to construct a succinct index.

Theorem 9 *A k-page graph G of n vertices and m edges, in which the edges are associated with σ labels in t pairs ($t \geq m$), can be represented using $k(n + o(n)) + \lg \binom{m\sigma}{t} + t \cdot o(\lg \sigma) + O(t)$ bits to support*:

- `edge_label` *in* $O(k)$ *time*;
- `lab_adjacency` *in* $O(\lg \lg \sigma + k)$ *time*;
- `lab_degree_edge` *in* $O(k(\lg \lg \lg \sigma)^2 \lg \lg \sigma)$ *time*;
- `lab_edges`(α, x) *in* $O(d(\lg \lg \lg \sigma)^2 \lg \lg \sigma + k)$ *time*, *where* $d =$ `lab_degree_edge`(α, x).

Alternatively, it can be represented using $n + o(n) + (2 + \epsilon)m(\lg k + o(\lg k)) + \lg \binom{m\sigma}{t} + t \cdot o(\lg \sigma) + O(t)$ bits, for any constant ϵ such that $0 < \epsilon < 1$, to support:

- `edge_label` *in* $O(1)$ *time*;
- `lab_adjacency` *in* $O(\lg \lg \sigma + \lg k)$ *time*;
- `lab_degree_edge` *in* $O((\lg \lg \lg \sigma)^2 \lg \lg \sigma)$ *time*;
- `lab_edges`(α, x) *in* $O(d(\lg \lg \lg \sigma)^2 \lg \lg \sigma)$ *time*, *where* $d =$ `lab_degree_edge`(α, x).

Proof To prove the first result, we use Lemma 16 to represent each page. Assume that m_i edges are embedded in the ith page, and that there are t_i edge-label pairs

between them and the alphabet set. The total space cost in bits is $k(n + o(n)) + \sum_{i=1}^{k} \left(\lg \binom{m_i\sigma}{t_i} + t_i \cdot o(\lg\sigma)\right) + O(t) \leq k(n + o(n)) + \lg \binom{m\sigma}{t} + t \cdot o(\lg\sigma) + O(t)$. To support the above operations on G, we perform them on each page. Note that to perform `lab_adjacency` on a page, it only takes constant time if the edge between these two vertices is not embedded in this page. It is then easy to show that the above running time of each operation is correct.

To prove the second result, we use the second result of Theorem 8 to encode the combinatorial structure of G. Recall that in its proof, we construct a multiple parenthesis sequence S, and two bit vectors B and B'. To encode the labels, we use an approach similar to that used in the proof of Lemma 16. We observe that the edges of G sorted by the positions of the corresponding opening parentheses (of any type) in S and the set of labels form a binary relation, and we denote this relation by R'. Similarly, the edges of G sorted by the positions of the corresponding closing parentheses (of any type) in S and the set of labels form a binary relation L'. We use Lemma 5 to represent the binary relation R' in $\lg \binom{m\sigma}{t} + t \cdot o(\lg\sigma) + O(t)$ bits. Observe that for the ith closing parenthesis in S, we can locate the position of the matching opening parenthesis using `m_match`, and compute the number of opening parentheses preceding it in S using B'. This can be performed in constant time. Thus we can use `object_select` on R' to directly support `object_select` on L'. We construct a succinct index of $t \cdot \lg\sigma$ bits using Lemma 4 for the binary relation L'. All these data structures occupy $n + o(n) + (2+\epsilon)m\lg k + m \cdot o(\lg k) + \lg \binom{m\sigma}{t} + t \cdot o(\lg\sigma) + O(t) = n + o(n) + (2+\epsilon)m(\lg k + o(\lg k)) + \lg \binom{m\sigma}{t} + t \cdot o(\lg\sigma) + O(t)$ bits, as $k \leq m$.

With the above data structures, the algorithms in the proof of Lemma 15 can be easily modified to support the operations on G, and the theorem follows.[5] □

As a planar graph can be embedded in at most 4 pages [30], we have the following corollary.

Corollary 5 *An edge-labeled planar graph of n vertices and m edges, in which the edges are associated with σ labels in t pairs ($t \geq m$), can be represented using $n + o(n) + \lg \binom{m\sigma}{t} + t \cdot o(\lg\sigma) + O(t)$ bits to support*:

- `edge_label` *in $O(1)$ time*;
- `lab_adjacency` *in $O(\lg\lg\sigma)$ time*;
- `lab_degree_edge` *in $O((\lg\lg\lg\sigma)^2 \lg\lg\sigma)$ time*;
- `lab_edges`(α, x) *in $O(d(\lg\lg\lg\sigma)^2 \lg\lg\sigma)$ time, where $d =$* `lab_degree_edge`(α, x).

Proof When we prove the second result in Theorem 9, we use the second result in Theorem 7 to encode the multiple parenthesis sequence S. Theorem 7 applies to the case in which the number of types of parentheses is non-constant. To prove this

[5] The first result of Theorem 8 can also be applied here using the same approach to achieve a different tradeoff. However, if we do so, the space cost will remain asymptotically the same, while the support for operations will be less efficient, as `m_match` cannot be performed in constant time. Thus we present only the better tradeoff using the second result of Theorem 8.

theorem, as a planar graph can be embedded in at most 4 pages, the number of type of parentheses in S is 4. Thus we can use Lemma 3 to represent S and the corollary directly follows. □

An alternative approach to obtain a similar result is to compute an embedding of the planar graph first, and then use Corollary 4 to represent it. The space cost is increased to $O(t+n) + \lg \binom{m\sigma}{t} + t \cdot o(\lg \sigma)$ bits. Thus Corollary 4 is more suitable when we need to support label-based rank/select operations in ccw order on edge labeled planar graphs that are already embedded in the plane.

6 Discussion

In this paper, we have presented a framework for succinct representation of properties of graphs in the form of labels. Our main results are the succinct representations of labeled and multi-labeled graphs (we consider vertex/edge labeled planar triangulations, vertex/edge labeled planar graphs, as well as edge labeled k-page graphs) to support various label queries efficiently. The additional space cost to store the labels is essentially the information-theoretic minimum. As far as we know, our representations are the first succinct representations of labeled graphs. We have also presented two preliminary results on unlabeled graphs to achieve the main results. First, we have designed a succinct representation of unlabeled planar triangulations and plane graphs to support the rank/select of edges in ccw (counter clockwise) order in addition to other operations supported in previous work [7–10]. Second, we have designed a succinct representation for a k-page graph when k is large to support various navigational operations more efficiently. In particular, we can test the adjacency of two vertices in $O(\lg k)$ time, while previous work uses $O(k)$ time [14, 22].

We expect that our approach can be extended to support some of the other types of graphs, which is an open research topic. Another open problem is to represent vertex labeled k-page graphs succinctly.

Our final comment is that because Theorems 2, 3, 5 and 6 provide succinct indexes for vertex/edge labeled planar triangulations and planar graphs, we can in fact store the labels in compressed form as Barbay *et al.* [2, 3] did to compress strings, while still providing the same support for operations. This also applies to Theorem 9, for which we apply succinct indexes for binary relations to encode the labels.

References

1. Barbay, J., Castelli Aleardi, L., He, M., Munro, J.I.: Succinct representation of labeled graphs. In: Proceedings of the 18th International Symposium on Algorithms and Computation. LNCS, vol. 4835, pp. 316–328. Springer, Berlin (2007)
2. Barbay, J., He, M., Munro, J.I., Rao, S.S.: Succinct indexes for strings, binary relations and multi-labeled trees. Manuscript http://www.cs.uwaterloo.ca/~mhe/research/manuscript/sisabr.pdf
3. Barbay, J., He, M., Munro, J.I., Rao, S.S.: Succinct indexes for strings, binary relations and multi-labeled trees. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 680–689 (2007)

4. Benoit, D., Demaine, E.D., Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Representing trees of higher degree. Algorithmica **43**(4), 275–292 (2005)
5. Bernhart, F., Kainen, P.C.: The book thickness of a graph. J. Combin. Theory, Ser. B **27**(3), 320–331 (1979)
6. Blandford, D.K., Blelloch, G.E., Kash, I.A.: Compact representations of separable graphs. In: Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 679–688 (2003)
7. Castelli Aleardi, L., Devillers, O., Schaeffer, G.: Succinct representation of triangulations with a boundary. In: Proceedings of the 9th Workshop on Algorithms and Data Structures. LNCS, vol. 3608, pp. 134–145. Springer, Berlin (2005)
8. Castelli Aleardi, L., Devillers, O., Schaeffer, G.: Succinct representations of planar maps. Theor. Comput. Sci., **408**(2–3), 174–187 (2008)
9. Chiang, Y.-T., Lin, C.-C., Lu, H.-I.: Orderly spanning trees with applications. SIAM J. Comput. **34**(4), 924–945 (2005)
10. Chuang, R.C.-N., Garg, A., He, X., Kao, M.-Y., Lu, H.-I.: Compact encodings of planar graphs via canonical orderings and multiple parentheses. In: Proceedings of the 25th International Colloquium on Automata, Languages and Programming, pp. 118–129 (1998)
11. Chung, F.R.K., Leighton, F.T., Rosenberg, A.L.: Embedding graphs in books: a layout problem with applications to VLSI design. SIAM J. Algebr. Discrete Methods **8**(1), 33–58 (1987)
12. Clark, D.R., Munro, J.I.: Efficient suffix trees on secondary storage. In: Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 383–391 (1996)
13. Farzan, A., Munro, J.I.: Succinct representations of arbitrary graphs. In: 16th Annual European Symposium on Algorithms, pp. 393–404 (2008)
14. Gavoille, C., Hanusse, N.: On compact encoding of pagenumber k graphs. Discrete Math. Theor. Comput. Sci. **10**(3), 23–34 (2008)
15. He, M.: Succinct indexes. PhD thesis, University of Waterloo, December (2007)
16. He, M., Munro, J.I., Rao, S.S.: Succinct ordinal trees based on tree covering. In: Proceedings of the 34st International Colloquium on Automata, Languages and Programming, pp. 509–520 (2007)
17. Isenburg, M., Snoeyink J.: Face fixer: compressing polygon meshes with properties. In: Proceedings of the 27th Annual Conference on Computer Graphics, pp. 263–270 (2000)
18. Jacobson G.: Space-efficient static trees and graphs. In: Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, pp. 549–554 (1989)
19. Jansson, J., Sadakane, K., Sung, W.-K.: Ultra-succinct representation of ordered trees. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 575–584 (2007)
20. Lipton, R.J., Tarjan, R.E.: A separator theorem for planar graphs. SIAM J. Appl. Math. **36**(2), 177–189 (1979)
21. Lu, H.-I., Yeh, C.-C.: Balanced parentheses strike back. ACM Trans. Algorithms **4**(3), 1–13 (2008)
22. Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. SIAM J. Comput. **31**(3), 762–776 (2001)
23. Munro, J.I., Rao, S.S.: Succinct representations of functions. In: Proceedings of the 31st International Colloquium on Automata, Languages and Programming, pp. 1006–1015 (2004)
24. Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Succinct representations of permutations. In: Proceedings of the 30th International Colloquium on Automata, Languages and Programming, pp. 345–356 (2003)
25. Raman, R., Raman, V., Satti, S.R.: Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. ACM Trans. Algorithms **3**(4), 43 (2007)
26. Rosenberg, A.L.: The DIOGENES design methodology: toward automatic physical layout. In: Proceedings of the International Workshop on Parallel Algorithms & Architectures, pp. 335–348 (1986)
27. Schnyder, W.: Embedding planar graphs on the grid. In: Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 138–148 (1990)
28. Tarjan, R.E.: Sorting using networks of queues and stacks. J. ACM **19**(2), 341–346 (1972)
29. Yamanaka, K., Nakano, S.-I.: A compact encoding of plane triangulations with efficient query supports. In: 2nd Annual Workshop on Algorithms and Computation, pp. 120–131 (2008)
30. Yannakakis, M.: Embedding planar graphs in four pages. J. Comput. Syst. Sci. **38**(1), 36–67 (1989)