Succinct Representation of Labeled Graphs[⋆]

Jérémy Barbay[1], Luca Castelli Aleardi[2], Meng He[1,3],
and J. Ian Munro[1]

[1] Cheriton School of Computer Science, University of Waterloo, Canada
{jbarbay,mhe,imunro}@uwaterloo.ca
[2] LIX (Ecole Polytechnique, France) and
CS Department (Université Libre de Bruxelles, Belgium)
amturing@lix.polytechnique.fr
[3] School of Computer Science, Carleton University, Canada
mhe@cg.scs.carleton.ca

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 vertex labeled planar triangulations, as well as edge labeled planar graphs and the more general 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 results. 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 graph when k is large to support various navigational operations more efficiently. In particular, we can test the adjacency of two vertices in $O(\lg k \lg\lg k)$ time, while previous work uses $O(k)$ time (10; 14).

1 Introduction

Graphs are fundamental combinatorial objects, widely used to represent various types of data. 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*, i.e. data structures that occupy space close to the information-theoretic lower bound while supporting efficient navigational operations, has been applied to various classes of graphs (5; 6; 7; 8; 12; 14).

Previous work focused on succinct graph representations which support efficiently testing the adjacency between two vertices and listing the edges incident

[⋆] This work was supported by NSERC of Canada, the Canada Research Chairs program, the IGM (University of Marne-la-Vallée) and the French "ACI Masses de données" program via the Geocomp project.

to a vertex (5; 6; 14). 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 (11). 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 efficiently support label-based connectivity queries, 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.[2].

We investigate three important classes of graphs: planar triangulations, planar graphs and k-page graphs. Planar graphs, in particular planar triangulations, correspond to the connectivity information underlying surface meshes. Triangle 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 (17), fault-tolerant processor arrays (15) and VLSI (9).

2 Preliminaries

2.1 Related Work

Jacobson (12) first proposed to represent unlabeled graphs succinctly. His approach is based on the concept of *book embedding* (4). 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, and listing the neighbors of a vertex in $O(d \lg n + k)$ bit probes, where d is the vertex degree.

Munro and Raman (14) improved his 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 (10) 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 (18), these results can be applied directly to planar graphs. In particular, a planar graph can be represented using $8n + 2m + o(n)$ bits to

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

[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.

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 in $O(d)$ time (14).

A different line of research is based on the canonical ordering of planar graphs. Chuang *et al.* (8) 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.* (7) further reduced the space cost to $2m + 3n + o(m + n)$ bits. When a planar graph is triangulated, Chuang *et al.* (8) showed how to represent it using $2m + 2n + o(m + n)$ bits.

Based on a partition algorithm, Castelli Aleardi *et al.* (5) 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.* (6) 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.

2.2 Multiple Parentheses

Chuang *et al.* (8) proposed succinct representation of *multiple parentheses*, a string of $O(1)$ types that may be unbalanced. Thus a multiple parenthesis sequence of p types of parentheses is a sequence over the alphabet $\{'(_1','(_2',...,'(_p',')_1',')_2',...,')_p'\}$. We call $'(_i'$ and $')_i'$ *type-i opening parenthesis* and *type-i closing parenthesis*, respectively. The operations considered are:

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

Chuang *et al.* (8) showed how to construct a $o(|S|)$-bit auxiliary data structure, for a string S of $O(1)$ types of parentheses stored explicitly, to support the above operations in constant time. We show how to improve this result in Corollary 1, and propose an encoding for the case when the number of types of parentheses is non-constant in Theorem 3.

2.3 Succinct Indexes for Binary Relations

Barbay *et al.* (2) 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.

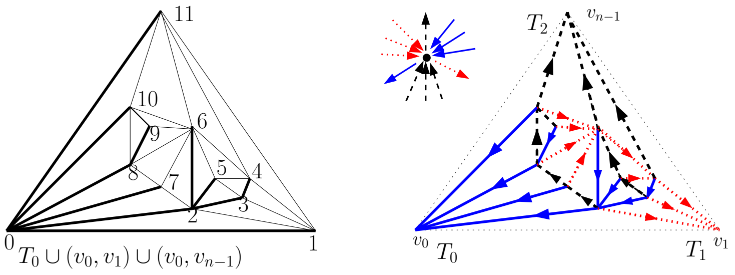


Fig. 1. A triangulated planar graph of 12 vertices with its canonical spanning tree $\overline{T}_0$ (on the left). On the right, it shows the triangulation induced with a realizer, as well as the local condition.

They considered sequences of n objects where each object can be associated with a subset of labels from $[\sigma]$, this association being defined by a binary relation of t pairs from $[n]\times[\sigma]$. The operations include: `object_access`(x, i), the i^{th} label associated with x in lexicographic order; `label_rank`(α, x), the number of objects labeled α up to (and including) x; `label_select`(α, r), the position of the r^{th} object labeled α; and `label_access`(x, α), whether object x is associated with label α. They defined the ADT through `object_access` and designed a succinct index of $t \cdot o(\lg \sigma)$ bits to support other operators efficiently.

2.4 Realizers and Planar Triangulations

A key notion in this paper is that of realizers of planar triangulations (see Figure 1 for an example).

Definition 1 (Schnyder (16)). *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 to v in 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.*

A fundamental property of realizers that we use extensively in Section 3 is:

Lemma 1 (Schnyder (16)). *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, T_1 and T_2 are spanning trees of $\mathcal{T} \setminus \{v_1, v_{n-1}\}$, $\mathcal{T} \setminus \{v_0, v_{n-1}\}$ and $\mathcal{T} \setminus \{v_0, v_1\}$, respectively.*

3 Vertex Labeled Planar Triangulations

3.1 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 its realizer by (T_0, T_1, T_2) following Lemma 1. By Lemma 1, T_0, T_1 and T_2 are three spanning trees of the internal nodes 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 $\mathcal{T}$ (8). In this section, we denote each vertex by its number in *canonical ordering*, which is the ccw preorder number in $\overline{T}_0$.

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* counter clockwise 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 vertex* 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 in cw order the contour of* $\overline{T_0}$*, the edges in* T_2 *incident to a node* v *are listed consecutively to induce an order on the vertices of* T_2*.*

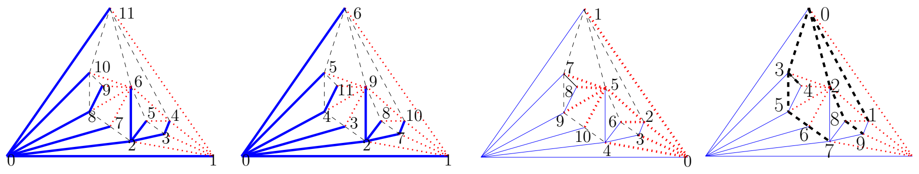
Note that the orders π_1 and π_2 do not correspond to previously studied traversal orders on the trees T_1 and T_2, as they are dependent on $\overline{T_0}$ through π_0 (see Figure 2). The following lemma is crucial (we omit the proof):

Lemma 2. *For any node* x*, its children in* T_1 *(or* T_2*), listed in ccw order (or cw order), have consecutive numbers in* π_1 *(or* π_2*). In the case of* $\overline{T_0}$*, the children of* x *are listed consecutively by a* `DFUDS` *(or* Depth First Unary Degree Sequence *(3)) traversal of* $\overline{T_0}$*.*

3.2 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 r^{th} 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.



π_0 $\quad$ $DFUDS\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 1 & 2 & 7 & 8 & 10 & 11 & 3 & 5 & 6 & 4 & 9 \end{bmatrix}$ $\quad$ $\pi_1\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 11 & 4 & 3 & 2 & 6 & 5 & 10 & 9 & 8 & 7 \end{bmatrix}$ $\quad$ $\pi_2\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 4 & 6 & 10 & 9 & 8 & 7 & 2 & 5 & 3 \end{bmatrix}$

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 of $\overline{T_0}$ are also shown.

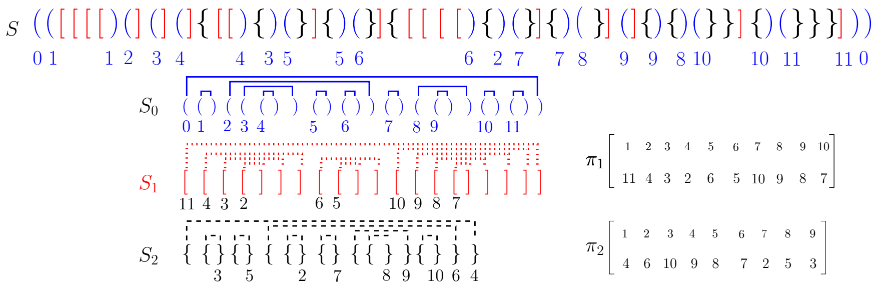


Fig. 3. The multiple parenthesis string encoding of the triangulation in Figure 2

– $\Pi_j(i)$, given the number of a node v_i in π_0 it returns the number of v_i in π_j;
– $\Pi_j^{-1}(i)$, given the number of a node v_i in π_j it returns its rank in π_0.

To represent a planar triangulation $\mathcal{T}$, we compute a realizer (T_0, T_1, T_2) of $\mathcal{T}$ following Lemma 1. We then encode the three trees T_0, T_1 and T_2 using a multiple parenthesis sequence S of length $2m$ consisting of three types of parenthesis. 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 first, second, and the third types of parentheses, respectively. We construct S as follows (see Figure 3 for an example).

Let $v_0, \ldots, v_{n-1}$ be the ccw preorder of the vertices of $\overline{T_0}$. Then the string S_0 is simply the balanced parenthesis encoding of the tree $\overline{T_0}$ (14): 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 in ccw order the edges incident to a vertex v_i, 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.

Thus S is of length $2m$, consisting of three types of parenthesis. It is easy to observe that the subsequences S_1 and S_2 are balanced parenthesis sequences of length $2(n-1)$ and $2(n-2)$, respectively.

We first observe some basic properties of the string S. Recall that a node v_i can be referred to by its preorder number in T_0, and by the position of the matching parenthesis pair $(_i$ and $)_i$ (let p_i and q_i denote their positions in S). Let be p_f (or q_l) be the position of the opening (or closing) parenthesis in S corresponding to the first (or last) child of node v_i in T_0.

Property 1. The following basic facts hold:
– Two nodes 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 T_0, T_1 or T_2;
– $p_i < p_f < q_l < q_i$;
– The number of edges incident to v_i and not belonging to the tree 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 correspond to the vertices v_i and the $')'$ at position q_l.

Observe that S_0 is the balanced parenthesis encoding of the tree $\overline{T_0}$ (14), so that if we store S_0 and construct the auxiliary data structures for S_0 as in (14), we can support a set of navigational operators on $\overline{T_0}$. S can be represented using the approach of Chuang *et al.* (8) (see Section 2.2) in $2m \lg 6 + o(m) = 2m\lceil \log_2 6\rceil + o(m) = 6m + o(m)$ bits. However, this encoding does not support the computation of an arbitrary word in S_0, so that 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 item $2m\lceil \log_2 6\rceil$ to $2m \log_2 6 + o(m)$, we have the following lemma (we omit the proof):

Lemma 3. *The string S can be stored in $2m \log_2 6 + o(m)$ bits to support the operators listed in Section 2.2 in constant time, as well as the computation of an arbitrary word, or $\Theta(n)$ bits of the balanced parenthesis sequence of $\overline{T_0}$.*

The same approach can be directly applied to a sequence of $O(1)$ types of parentheses:

Corollary 1. *Consider a multiple parenthesis sequence M of $2n$ parenthesis of p types, where $p = O(1)$. M can be stored using $2n \log(2p) + o(n)$ bits to support in $O(1)$ time the operators listed in Section 2.2, as well as the computation of an arbitrary word, or $\Theta(n)$ bits of the balanced parenthesis sequence of the parentheses of a given type in M.*

The following theorem shows how to support the navigational operations on triangulations. While the space used here is a little more than that of (7), the explicit use of the three parenthesis sequences seems crucial to exploiting the realizers to provide an efficient implementation supporting $\Pi_j(i)$ and $\Pi_j^{-1}(i)$.

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* `adjacency`*,* `degree`*,* `select_neighbor_ccw`*,* `rank_neighbor_ccw` *as well as the $\Pi_j(i)$ and $\Pi_j^{-1}(i)$ operators (for $j \in \{1,2\}$) 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 3. 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 (13; 14) to support the navigational operations on $\overline{T_0}$. As each vertex is denoted by its number in canonical ordering, vertex x corresponds to the x^{th} opening parenthesis in S_0. We now show that these structures are sufficient.

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 existing algorithms on S_0 (14). 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 y^{th} $'$($'$ in S, given by `m_first`$_{'[\,'}(S,$`m_select`$(S, y,'\,('))$, and compute the index, i, of its matching closing parenthesis, $'$[$'$, in S. We then check whether the nearest succeeding closing parenthesis $'$)$'$ of the $'$[$'$ retrieved, located using `m_first`$_{')'}(S, i)$, matches the x^{th} 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 of x in the trees $\overline{T_0}$, T_1 and T_2 (we denote the degree of a node in a tree as the number of nodes adjacent to it), 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 Chuang *et al.* (8). 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 x^{th} and the $x + 1^{\text{th}}$ $'$($'$ in S, and the indices, j_1 and j_2, of the $(n - x)^{\text{th}}$ and the $(n - x + 1)^{\text{th}}$ $'$)$'$ 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 when 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 i^{th} 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 (13). The children of x in T_1 corresponds to the parentheses $'['$ between the $(n-x)^{\text{th}}$ and the $(n-x+1)^{\text{th}}$ $')'$ in S, and because of 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. 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 i^{th} $'('$ in S, which is $\texttt{m\_select}(S, i, '(')$. We then locate the position, k, of the first $']'$ after position j, which is $\texttt{m\_first}_{']'}(S, j)$. After that, we locate the matching parenthesis of $S[j]$ using $\texttt{m\_match}(S, j)$ (p denotes the result). $S[p]$ is the parenthesis $'['$ that corresponds to the edge between u_i and its parent in T_1, and by the construction algorithm of S, the rank of $S[p]$ is the answer, which is $\texttt{m\_rank}(S, p, '[')$. The computation of Π_1^{-1} is exactly the inverse of the above process. Π_2 and Π_2^{-1} can be supported similarly. □

3.3 Vertex Labeled Planar Triangulations

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

– $\texttt{lab\_degree}(\alpha, x)$, the number of the neighbors of vertex x in G labeled α;
– $\texttt{lab\_select\_ccw}(\alpha, x, y, r)$, the r^{th} vertex labeled α among neighbors of vertex x after vertex y in ccw order, if y is a neighbor of x, and ∞ otherwise;
– $\texttt{lab\_rank\_ccw}(\alpha, x, y, z)$, the number of the neighbors of vertex x labeled α between y and z in ccw order if y and z are neighbors of x, and ∞ otherwise.

We define the interface of the ADT of labeled planar triangulations through $\texttt{node\_label}(v, i)$, which returns the i^{th} label associated to vertex v (i.e. the v^{th} vertex in canonical ordering).

Recall that Lemma 3 encodes the string S constructed in Section 3.2 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 of $\overline{T_0}$. We have the following lemma (we omit the proof).

Lemma 4. *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 in $O(1)$ time the operators listed in Section 2.2, as well as the computation of an arbitrary word, or $\Theta(n)$ bits of the balanced parenthesis sequence, and of the* `DFUDS` *sequence of $\overline{T_0}$.*

As Barbay *et al.* (2) did for multi-labeled trees, we now construct succinct indexes for vertex labeled planar triangulations. The main idea is to combine our succinct representation of planar triangulations with three instances of the succinct indexes for related binary relations:

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* `node_label` *in*

$f(n, \sigma, t)$ time on the vertices of $\mathcal{T}$, there is a succinct index using $t \cdot o(\lg \sigma)$ 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.*

To design a succinct representation of multi-labeled graphs using the above theorem, we use the approach of Barbay *et al.* (1) to encode R_0 using $t \lg \sigma + O(t)$ bits to support `object_access` in constant time, which directly supports `node_label` in $O(1)$ time. Thus:

Corollary 2. *A multi-labeled planar triangulation $\mathcal{T}$ of n vertices, associated with σ labels in t pairs ($t \geq n$) can be represented using $t \lg \sigma + t \cdot o(\lg \sigma)$ bits to support* `node_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.*

4 Edge Labeled Graphs with Pagenumber *k*

4.1 Multiple Parentheses

We now consider the succinct representation of multiple parenthesis sequences of p types of parentheses, where p is not a constant. We consider the following operations on a multiple parenthesis sequence $S[1..2n]$ in addition to those defined in Section 2.2: `m_rank`$'(S, i)$, the rank of the parenthesis at position i among parentheses of the same type in S; `m_findopen`(S, i) (`m_findclose`(S, i)), the matching closing (opening) parenthesis of the same type for the opening (closing) parenthesis at position i in S. Note that `m_findopen` and `m_findclose` are identical to the operator `m_match`. We define them here for the simplicity of the proofs of the theorems in this section. We have the following theorem (we omit all the proofs in this section because of space constraint):

Theorem 3. *A multiple parenthesis sequence of $2n$ parentheses of p types, in which the parentheses of any given type are balanced, can be represented using $2n \lg p + o(n \lg p)$ bits to support* `m_access`*,* `m_rank`$'$*,* `m_findopen` *and* `m_findclose` *in $O(\lg\lg p)$ time, and* `m_select` *in $O(1)$ time. Alternatively, $(2 + \epsilon)n \lg p + o(n \lg p)$ bits are sufficient to support these operations in $O(1)$ time, for any constant ϵ such that $0 < \epsilon < 1$.*

4.2 Graphs with Pagenumber *k* for Large *k*

In this section, on unlabeled graphs with page number k, we consider the operators `adjacency` and `degree` defined in Section 3.2, and the operator `neighbors`(x), returning the neighbors of x.

Previous results on succinctly representing k-page graphs (10; 14) support `adjacency` in $O(k)$ time. The lower-order term in the space cost of the result of Gavoille and Hanusse (10) is $o(km)$, which is dominant when k is large. Thus previous results mainly deal with the case when k is small. We consider large k.

Theorem 4. *A k-page graph of n vertices and m edges can be represented using $n + 2m \lg k + o(m \lg k)$ 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 $n + (2+\epsilon)m \lg k + o(m \lg k)$ 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 $0 < \epsilon < 1$.*

4.3 Edge Labeled Graphs with Pagenumber *k*

We consider the following operations on edge labeled graphs:

- `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 to vertex x that are labeled α;
- `lab_edges`(α, x), the edges incident to vertex x that are labeled α.

We first design succinct representation of edge labeled graphs with one page:

Lemma 5. *An outerplanar graph of n vertices and m edges in which the edges are associated with σ labels in t pairs ($t \geq n$) can be represented using $n + t(\lg \sigma + o(\lg \sigma))$ bits to support* `lab_adjacency` *and* `lab_degree_edge` *in $O(\lg\lg \sigma (\lg\lg\lg \sigma)^2)$ time, and* `lab_edges`(α, x) *in $O($*`lab_degree_edge`*$(\alpha, x) \lg\lg \sigma \lg\lg\lg \sigma)$ time.*

To support an edge labeled graph with k pages, we can use Lemma 5 to represent each page and combine all the pages to support navigational operations. Alternatively, we can use Theorem 4 and a similar approach to Lemma 5 to achieve a different tradeoff to improve the time efficiency for large k.

Theorem 5. *A k-page graph of n vertices and m edges in which the edges are associated with σ labels in t pairs ($t \geq n$) can be represented using $kn + t(\lg \sigma + o(\lg \sigma)$ bits to support* `lab_adjacency` *and* `lab_degree_edge` *in $O(k \lg\lg \sigma (\lg\lg\lg \sigma)^2)$ time, and* `lab_edges`(α, x) *in $O($*`lab_degree_edge`*$(\alpha, x) \lg\lg \sigma \lg\lg\lg \sigma + k)$ time. Alternatively, it can be represented using $n + (2m + \epsilon) \lg k + o(m \lg k) + m(\lg \sigma + o(\lg \sigma))$ bits to support* `lab_adjacency` *in $O(\lg k \lg\lg \sigma (\lg\lg\lg \sigma)^2)$ time,* `lab_degree_edge` *in $O(\lg\lg \sigma (\lg\lg\lg \sigma)^2)$ time, and* `lab_edges`(α, x) *in $O($*`lab_degree_edge`*$(\alpha, x) \lg\lg \sigma \lg\lg\lg \sigma)$ time, for any constant ϵ such that $0 < \epsilon < 1$.*

Corollary 3. *An edge-labeled planar graph of n vertices and m edges in which the edges are associated with σ labels in t pairs ($t \geq n$) can be represented using $4n + t(\lg \sigma + o(\lg \sigma))$ bits to support* `lab_adjacency` *and* `lab_degree_edge` *in $O(\lg\lg \sigma (\lg\lg\lg \sigma)^2)$ time, and* `lab_edges`(α, x) *in $O($*`lab_degree_edge`*$(\alpha, x) \lg\lg \sigma \lg\lg\lg \sigma)$ time.*

5 Concluding Remarks

In this paper, we present a framework of succinctly representing the properties of graphs in the form of labels. We expect that our approach can be extended to support other types of planar 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 Theorem 2 provides a succinct index for vertex labeled planar triangulations, we can in fact store the labels in compressed form as Barbay *et al.* (2) have done to compress strings, binary relations and multi-labeled trees, while still supporting the same operations. This also applies to Theorem 5, where we apply succinct indexes for binary relations.

References

[1] Barbay, J., Golynski, A., Munro, J.I., Rao, S.S.: Adaptive searching in succinctly encoded binary relations and tree-structured documents. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 24–35. Springer, Heidelberg (2006)

[2] 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. ACM Press, New York (2007)

[3] 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)

[4] Bernhart, F., Kainen, P.C.: The book thickness of a graph. Journal of Combinatorial Theory, Series B 27(3), 320–331 (1979)

[5] Castelli-Aleardi, L., Devillers, O., Schaeffer, G.: Succinct representation of triangulations with a boundary. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 134–145. Springer, Heidelberg (2005)

[6] Castelli-Aleardi, L., Devillers, O., Schaeffer, G.: Optimal succinct representations of planar maps. In: Proceedings of the 22nd ACM Annual Symposium on Computational Geometry, pp. 309–318 (2006)

[7] Chiang, Y.-T., Lin, C.-C., Lu, H.-I.: Orderly spanning trees with applications to graph encoding and graph drawing. In: Proceedings of the 12th Annual ACM-SIAM symposium on Discrete algorithms, pp. 506–515 (2001)

[8] 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)

[9] 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)

[10] Gavoille, C., Hanusse, N.: On compact encoding of pagenumber k graphs. Discrete Mathematics & Theoretical Computer Science (to appear, 2007)

[11] Isenburg, M., Snoeyink, J.: Face fixer: Compressing polygon meshes with properties. In: Proceedings of SIGGRAPH 2000, pp. 263–270 (2000)

[12] Jacobson, G.: Space-efficient static trees and graphs. In: Proceedings of the 30th Annual Symposium on Foundations of Computer Science, pp. 549–554 (1989)

[13] Lu, H.-I., Yeh, C.-C.: Balanced parentheses strike back. ACM Transactions on Algorithms (accepted, 2007)

[14] Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. SIAM J. Comput. 31(3), 762–776 (2001)
[15] Rosenberg, A.L.: The diogenes design methodology: toward automatic physical layout. In: Proceedings of the International Workshop on Parallel Algorithms & Architectures, pp. 335–348. North-Holland Publishing Co., Amsterdam (1986)
[16] Schnyder, W.: Embedding planar graphs on the grid. In: Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 138–148 (1990)
[17] Tarjan, R.E.: Sorting using networks of queues and stacks. J. Assoc. Comput. Mach. 19, 341–346 (1972)
[18] Yannakakis, M.: Four pages are necessary and sufficient for planar graphs. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Theory of Computing, pp. 104–108 (1986)