Orderly Spanning Trees with Applications to Graph Encoding and Graph Drawing*

Yi-Ting Chiang Ching-Chi Lin Hsueh-I Lu†

Institute of Information Science, Academia Sinica, Taipei 115, Taiwan

Abstract

The *canonical ordering* for triconnected planar graphs is a powerful method for designing graph algorithms. This paper introduces the *orderly pair* of connected planar graphs, which extends the concept of canonical ordering to planar graphs not required to be triconnected.

Let G be a connected planar graph. We give a linear-time algorithm that obtains an orderly pair (H, T) of G, where H is a planar embedding of G, and T is an *orderly spanning tree* of H. As applications, we show that the technique of orderly spanning trees yields (i) the best known encoding of G with query support, and (ii) the first area-optimal 2-visibility drawing of G.

1 Introduction

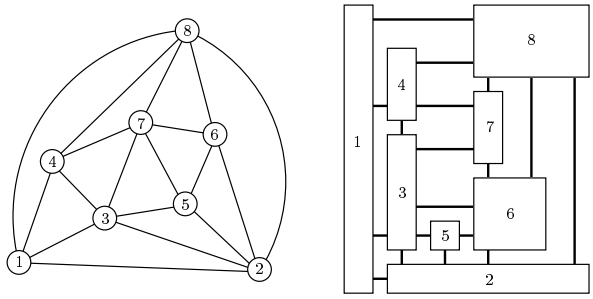
The *canonical orderings* of triconnected plane graphs [6, 10, 17, 18] are crucial in designing several graph-drawing and graph-encoding algorithms [3, 8, 11]. This paper introduces the concept of *orderly pair* for connected planar graphs, which extends that of canonical ordering to planar graphs not required to be triconnected. Let G be a connected planar graph. We give a linear-time algorithm that obtains an orderly pair (H, T) of G, where H is a planar embedding of G, and T is an *orderly spanning tree* of H.

As the first application of our orderly-pair algorithm, this paper deals with the problem of encoding a graph G into a binary string S with the requirement that S can be decoded to reconstruct G. This problem has been extensively studied with three objectives: (1) minimizing the length of S, (2) minimizing the time required to compute and decode S, and (3) supporting queries efficiently. As these objectives are often in conflict, a number of coding schemes with different trade-offs have been proposed in the literature. The widely useful adjacency-list encoding of an n-node m-edge graph G requires $2m\lceil \log_2 n \rceil$ bits. Talamo and Vocca [24] gave an encoding, obtainable in $O(n^3)$ time, that assigns an $O(d \log^3 n)$-bit label to each degree-d node. Without accounting for the time required to read the labels, the adjacency of two nodes can be answered from their encoding in $O(1)$ time. For certain graph families, Kannan, Naor and Rudich [16] gave schemes that encode each node with $O(\log n)$ bits and support $O(\log n)$-time testing of adjacency between two nodes. Cohen, Di Battista, Kanevsky, and Tamassia [5] gave a linear-space encoding of a k-connected G, obtainable in $O(n^4 m 4^k / k^2)$ time, from which one can determine in $O(1)$ time whether any two nodes are connected by $k+1$ node-disjoint paths. Jacobson [15] gave an $\Theta(n)$-bit encoding for a connected and simple planar G that supports traversal in $\Theta(\log n)$ time per node visited.

Under the *word model of computation* [2, 4, 9, 25, 26, 30], where operations such as read, write, and add on $O(\log n)$ consecutive bits take $O(1)$ time, an encoding S of G is *weakly convenient* [3] if it takes (a) $O(m+n)$ time to encode G and decode S; (b) $O(1)$ time to determine from S the adjacency of two nodes in G; and (c) $O(d)$ time to determine from S the neighbors of a degree-d node in G. If the degree of a node can be determined from a weakly convenient S in $O(1)$ times, then S is *convenient* [3]. For a planar G that may have multiple edges but no self-loops, Munro and Raman [20] gave the first nontrivial convenient encoding of G that has $2m + 8n + o(m+n)$ bits. Their result is based on the four-page decomposition of planar graphs [31] and an auxiliary string that encodes an involved three-level data structure for a string of parentheses. For a planar G that has (respectively, has no) multiple edges, Chuang, Garg, He, Kao, and Lu [3] improved the bit count to $2m + (5 + \frac{1}{k})n + o(m+n)$ (respectively, $\frac{5}{3}m + (5 + \frac{1}{k})n + o(n)$) for any positive constant k. They also gave a weakly convenient encoding of $2m + \frac{14}{3}n + o(m+n)$ (respectively, $\frac{4}{3}m + 5n + o(n)$) bits for a planar G that has (respectively, has no) multiple edges. In this paper, we give the best known convenient encodings for a planar G: If G may (respectively, may not) contain

*Research supported in part by NSC 89-2213-E-001-034 and NSC 89-2218-E-001-014.

†Corresponding author. Email: hil@iis.sinica.edu.tw. URL: http://www.iis.sinica.edu.tw/~hil.



(a) (b)

Figure 1: (a) A plane graph H. (b) A 2-visibility drawing of H.

multiple edges, then the bit count of our encoding is $2m+3n+o(m+n)$ (respectively, $2m+2n+o(n)$), which is even less than that of the weakly convenient encodings of Chuang *et al.* [3]. The bit counts are very close to Tutte's information-theoretical lower bound of roughly $3.58m$ bits for encoding connected plane graphs without any query support [29]. The bit count of our encoding for a planar G without multiple edges matches that of the best known convenient encoding for an outerplanar graph [20]. Besides the orderly-pair algorithm, our results are also based on an improved auxiliary string for a folklore encoding [3, 20] of a rooted tree T. With the auxiliary string of Munro and Raman [20], the degree of a degree-d node in T can be answered in $O(d)$ time. In this paper, we give a nontrivial auxiliary string that supports the degree query in $O(1)$ time.

Let H be an n-node m-edge simple plane graph. Let $v_1, v_2, \ldots, v_n$ be the nodes of H. A *2-visibility drawing* [8] of H consists of

- n rectangular boxes $b_1, b_2, \ldots, b_n$, and
- m horizontal and vertical lines segments that do not cross one another,

such that v_i and v_j are adjacent in H if and only if one of those m line segments connects b_i and b_j. For example, the picture shown in Figure 1(b) is a 2-visibility of the plane graph shown in Figure 1(a). Fößmeier, Kant, and Kaufmann [8] gave an $O(n)$-time algorithm that computes an $x \times y$ 2-visibility drawing of any n-node planar graph, where $x+y \leq 2n$. Moreover, they showed a planar graph whose $x \times y$ 2-visibility drawing has to satisfy $x + y \geq \frac{5}{3}n$, $x \geq \frac{2}{3}n$, and $y \geq \frac{2}{3}n$. As the second application of orderly spanning trees, we give the first known $n \times \frac{2}{3}n$ area-optimal 2-visibility drawing of a planar graph.

Related work. If one only need to reconstruct G with no query support, the code length can be substantially shortened. For this case, Turán [27] used $4m$ bits for a planar G that may have self-loops; this bound was improved by Keeler and Westbrook [19] to $3.58m$ bits. They also gave coding schemes for several important families of planar graphs. In particular, they used $1.53m$ bits for a triangulated simple G, and $3m$ bits for a connected G free of self-loops and degree-one nodes. For a simple triangulated (respectively, triconnected) G, He, Kao, and Lu [11] improved the bit count to $\frac{4}{3}m + O(1)$ (respectively, $\frac{3}{2}(\log_2 3)m + O(1)$). Rossignac [22] independently gave a $\frac{4}{3}m + O(1)$-bit encoding for plane triangulation. Although all these encodings can be encoded and decoded in linear time, none of them is known to be information-theoretic optimal. For example, the information-theoretic tight bound for plane triangulations, given by Tutte [28], is roughly $1.08m$. Recently, He, Kao, and Lu [12] proposed an $O(n \log n)$-time framework for encoding a graph in information-theoretically optimal number of bits. The framework is applicable to various classes of planar graphs. For labeled planar graphs, Itai and Rodeh [14] gave an encoding of $\frac{3}{2}n \log n + O(n)$ bits. For unlabeled general graphs, Naor [21] gave an encoding of $\frac{1}{2}n^2 - n \log n + O(n)$ bits.

The rest of the paper is organized as follows. Section 2 gives a polynomial-time algorithm for computing an orderly pair for any planar graph. Appendix A shows how to implement the algorithm to run in linear time. Section 3 describes several auxiliary strings for strings of parentheses. Section 4 shows how to design convenient encodings using orderly spanning trees. Section 5 shows how to compute an area-optimal 2-visibility drawing of a planar graph using orderly spanning trees. Section 6 concludes the paper.

2 Orderly spanning trees

Let H be a plane graph. Let T be a rooted spanning tree of H. Two nodes are *unrelated* if they are distinct and neither of them is an ancestor of the other in T. An edge of H is *unrelated* if its endpoints are unrelated. Let $v_1, v_2, \ldots, v_n$ be the counterclockwise preordering of the nodes in T. A node v_i is *orderly* in H with respect to T if the incident edges of v_i in H form the following four blocks in counterclockwise order around v_i:

B_1: edges incident to the parent of v_i;

B_2: unrelated edges incident to nodes v_j with $j < i$;

B_3: edges incident to the children of v_i; and

B_4: unrelated edges incident to nodes v_j with $j > i$,

where each block could be empty. T is an *orderly spanning tree* of H if (i) v_1 is on the boundary of the

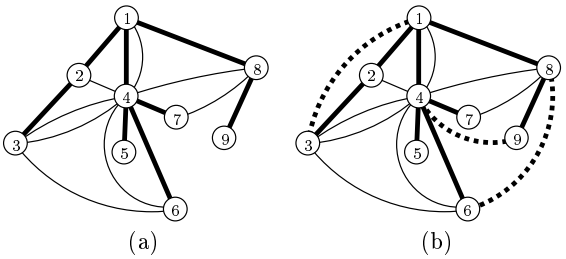


Figure 2: (a) A plane graph H. The thick edges form an orderly spanning tree T of H. The counterclockwise preordering of T is specified by the node labels. (b) Adding any of the dashed edges to $H - T$ would destroy the orderly pattern of some nodes, because v_1 and v_3 are related, the B_3 of v_4 is not supposed to contain any unrelated edge, and the B_4 of v_8 is not supposed to be incident to any lower-indexed node.

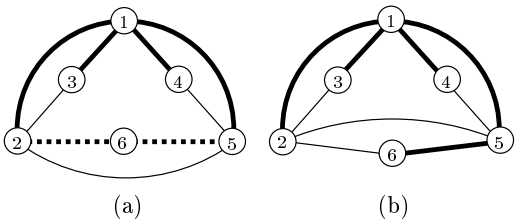


Figure 3: (a) A plane graph H that has no orderly spanning trees. (b) A different embedding of H that admits an orderly spanning tree consisting of the thick edges.

exterior face of H, and (ii) each node v_i is orderly in H. Clearly, if T is an orderly spanning tree of H, then each incident edge of v_1 in H belongs to T. In Figure 2(a), for example, one can easily verify that T is an orderly spanning tree of H. The dashed edges in Figure 2(b) violate the orderly property. In particular, v_1, v_3, v_4, and v_8 are not orderly in H with respect to T.

The concept of orderly pair originates from that of canonical spanning tree for triconnected plane graphs introduced by Chuang et al. [3]. More precisely, if H is a triconnected plane graph, then an orderly spanning tree of H is equivalent to a canonical spanning tree of H.

LEMMA 2.1. (SEE [3]) *An orderly spanning tree of any triconnected plane graph can be obtained in linear time.*

Not every connected plane graph admits an orderly spanning tree. For example, let H be the plane graph shown in Figure 3(a). If H had an orderly spanning tree T rooted at node 1, then the thick edges have to be in T, and thus the thin edges cannot be in T. Clearly, T contains exactly one of the dashed edges. In either case, however, the incident edges of the parent of node 6 is not orderly in H with respect to T. Since H is rotationally symmetric, H admits no orderly spanning trees. If we change the embedding of H by moving the edge $(2, 5)$ to the interior, as shown in Figure 3(b), then the new plane graph has an orderly spanning tree consisting of the thick edges rooted at node 1.

On the other hand, if we do not stick with a particular planar embedding of the given planar graph, then it is always possible to find a corresponding orderly spanning tree. Specifically, let G be a planar graph. We say that (H, T) is an *orderly pair* of G if (i) H is a planar embedding of G, and (ii) T is an orderly spanning tree of H. The rest of the section shows that an orderly pair of G can be found by a polynomial-time algorithm, whose linear-time implementation is given in Appendix A.

Let H be a plane graph that has no multiple edges. The *external boundary* of H is the boundary of the exterior face of H. A node is *external* in H if it is on the external boundary of H. An edge is *external* (respectively, *internal*) in H if it is (respectively, is not) on the external boundary of H. If v is an external node of a 2-connected H, then let $\mathrm{succ}(H, v)$ (respectively, $\mathrm{pred}(H, v)$) denote the counterclockwise (respectively, clockwise) neighbor of v on the external boundary of H. For example, in Figure 3(a), we have $\mathrm{succ}(H, 2) = 5$ and $\mathrm{pred}(H, 2) = 1$. An edge (u, v) of a 2-connected H is *mobile* if (i) both u and v are external in H, and (ii) there is a face F of H whose boundary contains u and v but not (u, v); that is, we can change the embedding of H by moving (u, v) into the interior of F. For example, $(1, 2)$, $(1, 5)$, and $(2, 5)$ are the mobile edges of the plane graph shown in Figure 3(b). Clearly, the endpoints of an internal mobile edge divides the external boundary into two segments. Since H has no multiple edges, each segment contains at least one node.

In order to obtain an orderly pair (H_0, T_0) of the input connected plane graph G_0 in polynomial time, we first compute an orderly pair (H_s, T_0) for G_s, where G_s is a simple plane graph obtained from G_0 by deleting all but one copy of each edge. If H_0 is obtained from H_s by adding back those edges in $G - G_s$ such that the multiple copies of each edge are bundled,[1] then (H_0, T_0) is clearly an orderly pair of G_0. It remains to focus on computing H_s and T_0.

The crucial step in our algorithm is a recursive subroutine $\mathsf{opair}(G, v_1)$ that returns an orderly pair (H, T) for a given 2-connected plane graph G and an external node v_1 of G such that the following properties

[1] For example, the two copies of the multiple edge $(3, 4)$ (respectively, $(4, 6)$) are (respectively, are not) bundled.

hold.

C1. Each external node of G remains external in H.

C2. If an external node v of H is not a leaf of T, then the rightmost child w of v in T is an external node of H such that $\mathrm{succ}(H, w) = v$.

Equipped with $\mathsf{opair}(G, v_1)$, we can compute H_0 and T_0 as follows. Let r_0 be an external node of G_s. First of all, we compute the 2-connected components $G_1, \ldots, G_k$ of G_s in $O(m+n)$ time. Let $\mathcal{T}$ be the tree of $G_1, \ldots, G_k$ rooted at a component containing r_0. For each $i = 1, 2, \ldots, k$, let r_i be the node of G_i that is closest to r_0 in G_s. Clearly, each r_i is an external node in G_i. Let $(H_i, T_i) = \mathsf{opair}(G_i, r_i)$. Let T_0 be the tree $\bigcup_{i=1}^{k} T_i$ rooted at r_0. Let H_0 be obtained by combining those H_i's in a bottom-up manner: suppose G_j is the parent of G_i in $\mathcal{T}$. Clearly, r_i is also in H_j. We place all the components in the subtree of $\mathcal{T}$ rooted at G_i into some face of H_j such that r_i is orderly in H_0 with respect to T_0. More precisely, the incident edges of r_i in H_i become part of the B_3 of r_i in H_0. Since each (H_i, T_i), with $1 \le i \le k$, is an orderly pair that satisfies Properties C1 and C2, the above algorithm is well defined. One can easily see that the running time is dominated by the overall time complexity of executing $\mathsf{opair}(G_i, r_i)$ for all indices j, since the rest of the algorithm clearly runs in $O(m+n)$ time.

The following definition is required to describe $\mathsf{opair}(G, v_1)$. Let v_1 be an external node of G. A node v of G is *stable* with respect to v_1 if (i) $v \notin \{v_1, \mathrm{succ}(G, v_1)\}$, (ii) v is not incident to any mobile edges of G, and (iii) $\mathrm{pred}(G, v)$ is in the 2-connected component of $G - \{v\}$ that contains $(v_1, \mathrm{succ}(G, v_1))$.

LEMMA 2.2. *If G has more than two nodes, then G has an embedding H that satisfies Property C1 and has a stable node with respect to v_1.*

Proof. Let H be obtained from G by iteratively moving external mobile edges into some interior faces until all mobile edges become internal. Since each step increases the length of the external boundary by at least one, the number of iterations is $O(n)$. Suppose $u_1, u_2, \ldots, u_\ell$, where $\ell \ge 3$ and $u_1 = v_1$, are the nodes on the external boundary of H in counterclockwise order. For notational convenience, let $u_{\ell+1} = v_1$. Assume for a contradiction that for each $i = 3, 4, \ldots, \ell$, u_i is not a stable node of H with respect to u_1. Note that $u_2 = \mathrm{pred}(H, u_3)$ is clearly in the 2-connected component of $H - \{u_3\}$ that contains (u_1, u_2). Since u_3 is not a stable node of H with respect to u_1, u_3 is incident to an internal mobile edge (u_3, u_{i_3}), where $3 < i_3 \le \ell + 1$. It follows that u_3 is in the 2-connected component of $H - \{u_4\}$ that contains (u_1, u_2). Since u_4 is not a stable node of H with respect to u_1, u_4 is incident to an internal mobile edge (u_4, u_{i_4}), where $4 < i_4 \le i_3$. By continuing the above argument, we know u_ℓ is incident to an internal mobile edge (u_ℓ, u_{i_ℓ}), where $\ell < i_\ell \le i_{\ell-1} \le i_{\ell-2} \le \cdots \le i_3 \le \ell + 1$. Thus we have $i_\ell = \ell + 1$, contradicting the fact that $(u_\ell, u_{\ell+1})$ is an external edge of H.

Now we define $\mathsf{opair}(G, v_1)$ as follows. If G has exactly two nodes, then $\mathsf{opair}(G, v_1)$ simply returns (H, T), where $H = T = G$. Otherwise, $\mathsf{opair}(G, v_1)$ performs the following steps.

1. Compute an embedding H of G as guaranteed in Lemma 2.2, and then replace G with H. Find a stable node v of G with respect to v_1. Let $v_2 = \mathrm{succ}(G, v_1)$. Then, compute the 2-connected components $G_1, \ldots, G_k$ of $G - \{v\}$. Let $\mathcal{T}'$ be the tree of $G_1, \ldots, G_k$ rooted at a component containing v_1.

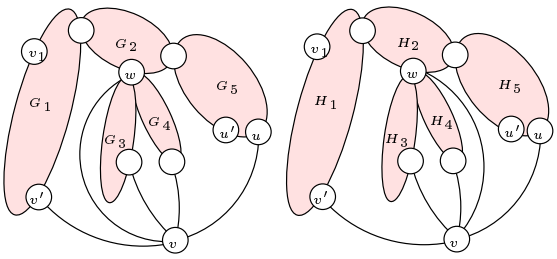
2. For each index $i = 1, 2, \ldots, k$, compute $(H_i, T_i) = \mathsf{opair}(G_i, r_i)$, where r_i is the node of G_i that is closest to v_1 in G. Since G_i and H_i satisfy Property C1 for each $i = 1, 2, \ldots, k$, we can glue all H_i's together according to $\mathcal{T}'$. Since G is a 2-connected plane graph, it is clear that, for any two 2-connected components of $G - \{v\}$, neither one is in the interior of the other in $G - \{v\}$. Let H' be the resulting plane graph. Let T' be the tree $\bigcup_{i=1}^{k} T_i$ rooted at v_1.

3. Each incident edge (v, w) of v in G is inserted to H' as follows. By Properties C1 and C2, v can be made the neighbor of w immediately succeeding the rightmost child, if any, of w in T' in counterclockwise order around w. Let H be the resulting plane graph. Let T be obtained from T' by inserting the edge $(v, \mathrm{succ}(H, v))$. Return (H, T).

An example is shown in Figure 4, where $G_1, \ldots, G_5$ are the 2-connected components of $G - \{v\}$. Clearly, v is a stable node of G with respect to v_1. The subroutine $\mathsf{opair}(G, v_1)$ computes (H_i, T_i) for each $i = 1, 2, \ldots, 5$, and then glues all H_i's together according the tree structure $\mathcal{T}'$ of $G_1, \ldots, G_5$. When adding back the edge (v, w) to H', the algorithm moves it to the right of H_4, since the rightmost child of w in T' is in H_4. The parent of v in T is u.

LEMMA 2.3. *The output (H, T) of $\mathsf{opair}(G, v_1)$ is an orderly pair of G with respect to v_1 such that Properties C1 and C2 hold.*

Proof. We prove the lemma by induction on the number of nodes in G. The lemma holds trivially when G



(a) G (b) H

Figure 4: An example illustrating $\mathsf{opair}(G, v_1)$.

has two nodes. Suppose G has more than two nodes. We first show that T' is an orderly spanning tree of H'. Clearly, it suffices to verify that each r_i, with $2 \leq i \leq k$, is indeed orderly in H' with respect to T'. Let $H_{i_0}, H_{i_1}, \ldots, H_{i_\ell}$ be the components of H' that contains r_i, where H_{i_0} is the parent of H_i in $\mathcal{T}'$. Clearly, $r_{i_j} = r_i$ holds for each $j = 1, 2, \ldots, \ell$. By the inductive hypothesis, we know that Property C2 holds for (G_{i_0}, H_{i_0}). Also, for each $j = 1, 2, \ldots, \ell$, the incident edges of r_i in H_{i_j} belongs to T_{i_j} and T'. Thus, one can easily verify that r_i is orderly in H' with respect to T'.

Now we show that all the required properties hold for H and T. By the choice of the parent u of v in T, it is clear that for each neighbor w of v in H, the index of v in T is higher than that of w. Since T' is an orderly spanning tree of H', it follows from the definition of H that each neighbor w of v in H is orderly in H with respect to T. Clearly, u is also orderly in H with respect to T. Therefore, T is indeed an orderly spanning tree of H. It is not difficult to see that Property C1 holds for (G, H). It remains to ensure Property C2 for H and T. Clearly, it suffices to prove the property for $\mathrm{pred}(H, u)$ and $\mathrm{succ}(H, u)$ of H, since u is a leaf of T, and the incident edges of the other external nodes of H are the same as those in H'. Since $(\mathrm{pred}(H, u), u)$ is an edge in $H - T$, the property of $\mathrm{pred}(H, u)$ in H with respect to T follows from that in H' with respect to T'. Similarly, since u is the rightmost child of $\mathrm{succ}(H, u)$ in T, the property of $\mathrm{succ}(H, u)$ in H with respect to T follows from that in H' with respect to T'.

A naive implementation of the above algorithm runs in $O(n^2)$ time. Since the time complexity of our algorithm is clearly dominated by that of opair, we prove the next theorem in Appendix A by giving a linear-time implementation of opair.

THEOREM 2.1. *Given any n-node m-edge connected planar graph G that has no self-loops, it takes $O(m+n)$ time to compute an orderly pair of G.*

3 Auxiliary strings for strings of parentheses

In this section we give the data structures required by the convenient encodings to be described in Section 4.

Let $|S|$ denote the length of a string S. Clearly, an S consisting of t distinct symbols can be encoded in $|S|\lceil \log_2 t \rceil$ bits. For example, if S consists of parentheses and brackets, including open and close ones, then S can be encoded in $2|S|$ bits. S is *binary* if it consists of at most two distinct symbols. For each $1 \leq i \leq j \leq |S|$, let $S[i, j]$ be the length-$(j - i + 1)$ substring of S from the i-th position to the j-th position. Define $S[i] = S[i, i]$. For each $i \leq j < i \leq |S|$, let $S[i, j]$ be the empty string. $S[k]$ is *enclosed* by $S[i]$ and $S[j]$ in S if $i < k < j$. Let $\mathrm{select}(S, i, \square)$ be the position of the i-th $\square$ in S. Let $\mathrm{rank}(S, k, \square)$ be the number of $\square$'s before or at the k-th position of S. Clearly, if $k = \mathrm{select}(S, i, \square)$, then $i = \mathrm{rank}(S, k, \square)$.

An *auxiliary string* χ of S is a binary string with $|\chi| = o(|S|)$ that is obtainable from S in $O(|S|)$ time.

LEMMA 3.1. (SEE [1, 7]) *For any strings $S_1, S_2, \ldots, S_k$ with $k = O(1)$, there is an auxiliary string χ_0 such that given the concatenation of $\chi_0, S_1, S_2, \ldots, S_k$ as input, the index of the first symbol of any given S_i in the concatenation can be computed in $O(1)$ time.*

Let $S_1 + S_2 + \cdots + S_k$ denote the concatenation of $\chi_0, S_1, S_2, \ldots, S_k$ as in Fact 3.1.

Suppose S is a string of multiple types of parentheses. Let $\mathrm{rev}(S)$ be the string R such that $R[i]$ is the opposite type of parenthesis $S[|S| + 1 - i]$. For example, $\mathrm{rev}(\text{“)()])[”}) = \text{“]([()(”}$. For an open parenthesis $S[i]$ and a close one $S[j]$ of the same type where $i < j$, the two *match* in S if every parenthesis of the same type that is enclosed by them matches one enclosed by them. S is *balanced in type k* if every parenthesis of type k in S belongs to a matching parenthesis pair. S is *balanced* if S is empty or is balanced in all types of parentheses. Here are some queries defined for any balanced S. Let $\mathrm{match}(S, i)$ be the position of the parenthesis in S that matches $S[i]$. Let $\mathrm{enclose}_k(S, i_1, i_2)$ be the position pair (j_1, j_2) of the closest matching parenthesis pair of the k-th type that encloses $S[i_1]$ and $S[i_2]$.

LEMMA 3.2. (SEE [3, 20]) *For any balanced string S of $O(1)$ types of parentheses, there is an auxiliary string $\chi_1(S)$ such that each of $\mathrm{rank}(S, i, \square)$, $\mathrm{select}(S, i, \square)$, $\mathrm{match}(S, i)$, and $\mathrm{enclose}_k(S, i, j)$ can be determined from $S + \chi_1(S)$ in $O(1)$ time.*

For a string S of parentheses that may be unbalanced, if $S[i]$ is an unmatched open parenthesis, then let $\mathrm{match}(S, i) = |S| + 1$. Now we define the function $\mathrm{nextlevel}(S, i)$. Suppose $S[i]$ is of type k. Roughly

speaking, $\mathrm{nextlevel}(S,i)$ is the number of parentheses $S[j]$ of type k satisfying $\mathrm{enclose}_k(S,j,\mathrm{match}(S,j)) = (i,\mathrm{match}(S,i))$. More rigorously, $\mathrm{nextlevel}(S,i)$ is the number of type-k parentheses $S[j]$ with $i+1 \le j \le \mathrm{match}(S,i)-1$ such that if $S[j]$ is open, then $S[i+1,j-1]$ is balanced in type k; otherwise $S[i+1,j]$ is balanced in type k. If $S[i]$ is closed, then we define $\mathrm{nextlevel}(S,i)$ to be $\mathrm{nextlevel}(\mathrm{rev}(S),|S|+1-\mathrm{match}(S,i))$. Therefore, if

$$S = \texttt{(((()[[[)[(]]]()(])()[)[(]]()))}, \tag{3.1}$$

then we have $\mathrm{nextlevel}(S,1) = 6$ and $\mathrm{nextlevel}(S,8) = 2$. Clearly, if S is balanced, then $\mathrm{nextlevel}(S,i)$ is even for each i. The next lemma extends the set of queries supported in Fact 3.2.

LEMMA 3.3. *For any balanced string S of $O(1)$ types of parentheses, there is an auxiliary string $\chi_2(S)$ such that nextlevel(S,i) can be answered from $S+\chi_2(S)$ in $O(1)$ time.*

Proof. Let $s = |S|$. Let t be the number of distinct types of parentheses in S. Let b the smallest integer such that $2t \le 2^b$. Clearly, each symbol of S can be encoded in b bits. By $t = O(1)$, we have $b = O(1)$. Let $\ell = \lfloor \frac{1}{2}\log_{2^b} s \rfloor$. Clearly, any substring $S[i,j]$ with $j \le i+\ell-1$ has $O(\sqrt{s})$ possible distinct values. Define tables M_1 and M_2 for S by letting $M_1[S[i,i+\ell-1]] = \mathrm{nextlevel}(S[i,i+\ell-1],1)$ and $M_2[S[i,j]] = \mathrm{nextlevel}(\mathrm{rev}(S[i,j]),1)$, for any i,j with $1 \le i \le j \le i+\ell-1$. For each $k = 1,2,\ldots,t$, define tables M_3^k and M_4^k as follows. For each $i = 1,2,\ldots,\lceil \frac{s}{\ell^2} \rceil$, let $M_3^k[i] = (j,c)$, where $c = \mathrm{nextlevel}(S,j)$ and j is the largest index such that $j \le i\ell^2$, $\mathrm{nextlevel}(S,j) > \ell^2$, and $S[j]$ is a close parenthesis of type k. Let $M_4^k[i] = (j,c)$, where $c = \mathrm{nextlevel}(S,j)$ and j is the smallest index such that $j \ge i\ell^2$, $\mathrm{nextlevel}(S,j) > \ell^2$, and $S[j]$ is an open parenthesis of type k.

An open $S[i]$ is *special* if (i) $\mathrm{match}(S,i)-i > \ell$, (ii) $\mathrm{nextlevel}(S,i) \le \ell^2$, and (iii) for each $S[j]$ with $j > i$ and $S[j] = S[i]$, we have $j-i > \ell$ or $\mathrm{match}(S,i) - \mathrm{match}(S,j) > \ell$. A close $S[i]$ is *special* if $S[\mathrm{match}(S,i)]$ is special. For each $k = 1,2,\ldots,t$, define tables M_5^k and M_6^k as follows. For each $i = 1,2,\ldots,\lceil \frac{s}{\ell} \rceil$, let $M_5^k[i] = (j,c)$, where $c = \mathrm{nextlevel}(S,j)$ and j is the largest index such that $j \le i\ell$ and $S[j]$ is a special close parenthesis of type k. Let $M_6^k[i] = (j,c)$, where $c = \mathrm{nextlevel}(S,j)$ and j is the smallest index such that $j \ge i\ell$ and $S[j]$ is a special open parenthesis of type k. Let $\chi_2(S) = M_1 + M_2 + M_3^1 + M_4^1 + M_5^1 + M_6^1 + \cdots + M_3^t + M_4^t + M_5^t + M_6^t$. It follows from $t = O(1)$ that $|\chi_2(S)| = o(s)$. It remains to show that $\mathrm{nextlevel}(S,i)$ can be answered from S and $\chi_2(S)$ by the algorithm shown in Figure 5, which clearly runs in $O(1)$ time.

function $\mathsf{nextlevel}(S,i)\{$
1. let k, with $1 \le k \le t$, be the type of $S[i]$;
2. let $i_1 = \min\{i, \mathrm{match}(S,i)\}$;
3. let $i_2 = \mathrm{match}(S,i_1)$;
4. let $(j,c) = M_3^k[\lceil \frac{i_2}{\ell^2} \rceil]$; if $j = i_2$ then return c;
5. let $(j,c) = M_4^k[\lfloor \frac{i_1}{\ell^2} \rfloor]$; if $j = i_1$ then return c;
6. let $(j,c) = M_5^k[\lceil \frac{i_2}{\ell} \rceil]$; if $j = i_2$ then return c;
7. let $(j,c) = M_6^k[\lfloor \frac{i_1}{\ell} \rfloor]$; if $j = i_1$ then return c;
8. let $j_1 = i_1 + \ell - 1$;
9. if $i_2 - i_1 \le 2\ell$ then
   let $j_2 = i_1 + \ell$;
   else
   let $j_2 = i_2 - \ell + 1$;
10. return $M_1[S[i_1,j_1]] + M_2[S[j_2,i_2]]$;

$\}$

Figure 5: An $O(1)$-time algorithm $\mathsf{nextlevel}(S,i)$.

By the definitions of $M_3^k, M_4^k, M_5^k, M_6^k$, if a value c is returned by Steps 4–7, then $c = \mathrm{nextlevel}(S,i)$.

The rest of the proof assumes that Step 8 is executed. We first show that $S[i]$ is not special and $\mathrm{nextlevel}(S,i) \le \ell^2$ holds. Assume for a contradiction that $S[i]$ is special. By the definitions of M_5^k and M_6^k, there exists an index j such that (a) $S[j] = S[i]$; (b) $S[j]$ and $S[\mathrm{match}(S,j)]$ encloses $S[i]$ and $S[\mathrm{match}(S,i)]$; (c) $S[j]$ is special; (d) $1 \le |j-i| < \ell$; and (e) $1 \le |\mathrm{match}(S,j) - \mathrm{match}(S,i)| < \ell$. By the definition of special parentheses, Condition (c) contradicts Conditions (d) and (e). Assume for a contradiction that $\mathrm{nextlevel}(S,i) > \ell^2$ holds. By the definitions of M_3^k and M_4^k, there exists an index j such that (a) $S[j] = S[i]$; (b) $S[j]$ and $S[\mathrm{match}(S,j)]$ encloses $S[i]$ and $S[\mathrm{match}(S,i)]$; (c) $\mathrm{nextlevel}(S,j) > \ell^2$; (d) $1 \le |j-i| < \ell^2$; and (e) $1 \le |\mathrm{match}(S,j) - \mathrm{match}(S,i)| < \ell^2$. By Conditions (d) and (e), we know $\mathrm{nextlevel}(S,j) \le \frac{1}{2}(|j-i| - 1 + |\mathrm{match}(S,j) - \mathrm{match}(S,i)| - 1) + 1 < \ell^2$, contradicting Condition (c).

Now we are ready to argue that the algorithm correctly returns $\mathrm{nextlevel}(S,i)$ in Step 10. By Steps 2 and 3, $S[i_1]$ is open and $S[i_2]$ is closed. By Steps 8 and 9, we know $j_1 < j_2$. If $i_2 - i_1 \le 2\ell$, then $M_1[S[i_1,j_1]] + M_2[S[j_2,i_2]] = \mathrm{nextlevel}(S[i_1,j_1],1) + \mathrm{nextlevel}(\mathrm{rev}(S[j_1+1,i_2]),1) = \mathrm{nextlevel}(S,i)$. Now we assume $i_2 - i_1 > 2\ell$. Since $S[i_1]$ is not special and $\mathrm{nextlevel}(S,i_1) \le \ell^2$, by the definition of special parentheses, there exists an index j_1' with $S[j_1'] = S[i_1]$, $0 < j_1' - i_1 \le \ell$, and $0 < i_2 - j_2' \le \ell$, where $j_2' = \mathrm{match}(S,j_1')$. Therefore, it is not difficult to see that $M_1[S[i_1,j_1]] + M_2[S[j_2,i_2]] = \mathrm{nextlevel}(S[i_1,j_1],1) + \mathrm{nextlevel}(\mathrm{rev}(S[j_2,i_2]),1) = \mathrm{nextlevel}(S[i_1,j_1'],1) + \mathrm{nextlevel}(\mathrm{rev}(S[j_2',i_2]),1) = \mathrm{nextlevel}(S,i)$, proving the

lemma.

4 Convenient encodings

In this section we give the best known convenient encodings for planar graphs as the first application of our orderly-pair algorithm.

A folklore encoding [3, 11, 20] S of an n-node simple rooted tree T is a balanced string of $2n$ parentheses that represents a counterclockwise depth-first traversal of T. Initially, an open (respectively, close) parenthesis denotes a descending (respectively, ascending) edge traversal. Then this string is enclosed by an additional matching parenthesis pair. For example, the string in Equation (4.2) is the folklore encoding for the tree T in Figure 2. Clearly, the i-th node in the counterclockwise depth-first traversal corresponds to the matching parentheses pair at $S[\text{select}(S, i, ()]$ and $S[\text{match}(\text{select}(S, i, ()))]$.

Let H be an n-node connected plane graph that may have multiple edges but no self-loops. Let T be a spanning tree of H rooted at v_1. Let $v_1 v_2 \cdots v_n$ be a counterclockwise preordering of T. Let degree(i) be the number of edges incident to v_i in H. Let children(i) be the number of children of v_i in T. Let above(i) (respectively, below(i)) be the number of edges (v_i, v_j) of H such that v_j is the parent (respectively, a child) of v_i in T. Let low(i) (respectively, high(i)) be the number of edges (v_i, v_j) of H such that $j < i$ (respectively, $j > i$) and v_j is neither the parent nor a child of v_i in T. In Figure 2(a), for instance, we have above(4) = 2, below(4) = 4, low(4) = 3, and high(4) = 2. Clearly, degree(i) = above(i) + below(i) + low(i) + high(i). If H has no multiple edges, then clearly below(i) = children(i).

The *T-code* of H is a triple (S_1, S_2, S_3) of binary strings, where S_1, S_2, and S_3 are defined as follows.

- S_1 is the folklore encoding of T.
- Let $p_i = \text{select}(S_1, i, ()$ and $q_i = \text{match}(S_1, p_i)$. S_2 has exactly $2n$ copies of 1, in which low(i) copies of 0 immediately succeeds the p_i-th 1, and high(i) copies of 0 immediately succeeds the q_i-th 1.
- S_3 has exactly n copies of 1, where above(i) + below(i) − children(i) $-\delta_{i\geq 2}$ copies of 0 immediately succeeds the i-th 1.

For example, if H and T are as shown in Figure 2(a), then

$$
\begin{aligned}
(4.2)\quad S_1 &= \texttt{((())(()()())(()));}\\
S_2 &= \texttt{1111000101000111011101010011 11;}\\
S_3 &= \texttt{1011100110111.}
\end{aligned}
$$

Clearly $|S_1| = 2n$, $|S_2| = 2n + \sum_{i=1}^{n} \text{low}(i) + \text{high}(i)$, and $|S_3| = \sum_{i=1}^{n} (\text{above}(i) + \text{below}(i) - \text{children}(i)) + 1$. Therefore $|S_1| + |S_2| + |S_3| = 2m + 3n + 2$. Moreover, if H has no multiple edges, then $|S_3| = n$ and thus $|S_1| + |S_2| = 2m + 2n + 2$.

The next theorem gives our convenient encodings.

THEOREM 4.1. *Let G be an input n-node m-edge planar graph having no self-loops. If G has (respectively, has no) multiple edges, then G has a convenient encoding, obtainable in $O(m+n)$ time, that has $2m+3n+o(m+n)$ (respectively, $2m+2n+o(n)$) bits.*

Proof. (sketch) The techniques used in the proof are mostly adapted from [3]. We focus on the case that G is connected. It is not difficult to remove this restriction. Let H and T be as guaranteed by Theorem 2.1. Let (S_1, S_2, S_3) be the T-code of H. We prove that there exists an $o(m+n)$-bit string χ, obtainable in $O(m+n)$ time, such that $S_1 + S_2 + S_3 + \chi$ is a convenient encoding of G. Clearly, if G has no multiple edges, then S_3 consists of n copies of 1, and thus $S_1 + S_2 + \chi$ suffices.

One can verify that $\text{low}(i) = \text{select}(S_2, p_i + 1, 1) - \text{select}(S_2, p_i, 1) - 1$ and $\text{high}(i) = \text{select}(S_2, q_i + 1, 1) - \text{select}(S_2, q_i, 1) - 1$, where $p_i = \text{select}(S_1, i, ()$ and $q_i = \text{match}(S_1, p_i)$. It is also clear that $\text{children}(i) = \text{nextlevel}(S_1, p_i)/2$. By the definition of S_3, we know $\text{above}(i) + \text{below}(i) - \text{children}(i) = \text{select}(S_3, i+1, 1) - \text{select}(S_3, i, 1) - 1 + \delta_{i\geq 2}$. Let $\chi' = \chi_1(S_1) + \chi_1(S_2) + \chi_1(S_3) + \chi_2(S_1)$. By $\text{degree}(i) = \text{above}(i) + \text{below}(i) + \text{low}(i) + \text{high}(i)$, Fact 3.2, and Lemma 3.3, we know that degree(i) can be computed from $S_1 + S_2 + S_3 + \chi'$ in $O(1)$ time.

Let S be the string of two types of parentheses obtained from S_1 and S_2 as follows. Let (and) be of type 1. Let [and] be of type 2. Initially, for each $i = 1, 2, \ldots, 2n$, replace the i-th 1 of S_2 by $S_1[i]$. Then, replace each 0 of S_2 by a bracket such that the bracket is open if and only if the last parenthesis in S that precedes this 0 is closed. More precisely, for each $i = 1, 2, \ldots, |S_2|$, let

$$
S[i] = \begin{cases} S_1[j_1] & \text{if } S_2[i] = 1; \\ \texttt{]} & \text{if } S_2[i] = 0 \text{ and } S_1[j_i] = \texttt{(}; \\ \texttt{[} & \text{if } S_2[i] = 0 \text{ and } S_1[j_i] = \texttt{)}, \end{cases}
$$

where $j_i = \text{rank}(S_2, i, 1)$. For example, if H and T are as given in Figure 2(a), then S is as in Equation (3.1). It is not difficult to prove that there exists an auxiliary string χ_3, similar to the M_1 in the proof of Lemma 3.3, such that any $O(\log n)$ consecutive symbols of S can be obtained from $S_1 + S_2 + \chi_3$ in $O(1)$ time.

For each $i = 1, 2, \ldots, n$, let L_i be the interval $[\ell_i + 1, \text{select}(S_2, \text{rank}(S_2, \ell_i, 1) + 1, 1) - 1]$ and R_i be the interval $[h_i + 1, \text{select}(S_2, \text{rank}(S_2, h_i, 1) + 1, 1) - 1]$,

where $\ell_i = \text{select}(S, i, ()$ and $h_i = \text{match}(S, \ell_i)$. Since T is an orderly spanning tree of H, it is not hard to see that $h_i < h_{i'} < \ell_{j'} < \ell_j$ holds for any two unrelated edges (v_i, v_j) and $(v_{i'}, v_{j'})$, with $i < j$ and $i' < j'$, such that $(v_{i'}, v_{j'})$ is enclosed by $T \cup (v_i, v_j)$. It follows that v_i and v_j, with $i < j$, are adjacent in $H - T$ if and only if there exists an index $\ell \in R_i$ such that $\text{match}(S, \ell) \in L_j$. Therefore, one can determine whether (v_i, v_j) is an unrelated edge of H by checking whether $i'' \in R_i$ and $j'' \in L_j$ hold, where $(i'', j'') = \text{enclose}_2(S, \text{select}(S, \text{rank}(S_2, h_i, 1) + 1, (), \ell_j)$. Therefore, the adjacency query can be answered from $S_2 + S + \chi_1(S_2) + \chi_1(S)$ in $O(1)$ time.

It is not difficult to show that the neighbors of a degree-d node can be listed from $S + \chi_1(S)$ in $O(d)$ time. It is also not difficult to show that G can be reconstructed from χ. Therefore the theorem is proved by letting $\chi = \chi' + \chi_3$.

5 Optimal 2-visibility drawings

In this section we give the area-optimal $n \times \frac{2}{3}n$ 2-visibility drawing for planar graphs as the second application of the concept of orderly spanning trees. For counting the area, suppose the corner coordinates of the boxes are integers, and each box has size no less than 1×1. Let the edges be placed at half-integer coordinates.

LEMMA 5.1. *If we are given an orderly spanning tree of H with ℓ leaves, then H has an $n \times \ell$ 2-visibility drawing obtainable in $O(n)$ time.*

Proof. (sketch) Let $v_1, v_2, \ldots, v_n$ be the counterclockwise preordering of the given orderly spanning tree T of H. For each $i = 1, 2, \ldots, n$, let H_k (respectively, T_k) be the subgraph of H (respectively, T) induced by $v_1, v_2, \ldots, v_i$. Let H'_i be the plane graph obtained from H_i by adding all the edges (v_j, v_i) not in G_k, where (a) $j < i$, (b) v_j and v_i are unrelated in T_i, and (c) v_j is on the external boundary of H_i. Clearly, T_i rooted at v_1 is an orderly spanning tree of H_i and H'_i. Also, v_i is always a leaf of T_i. Since H_i is a subgraph of H'_i, it suffices to prove by induction on i that H'_i admits an $i \times \ell_i$ 2-visibility drawing, where ℓ_i is the number of leaves in T_i. The induction basis holds trivially. Assume that H'_i admits an $i \times \ell_i$ 2-visibility drawing. In order to draw the v_{i+1} and its incidental edges to it unrelated nodes in H'_{i+1}, we need only to extend b_i and some b_j, with $j < i$, vertically down by one unit and then split the extended b_i into two new boxes b_i and b_{i+1}. To draw the vertical line representing the edge of H'_{i+1} incident to v_{i+1} and its parent v_p in T, we need to extend b_{i+1} and b_p horizontally to the right by one unit only if v_i remains a leaf in T_{i+1}. Therefore H'_{i+1} admits an $(i+1) \times \ell_{i+1}$ 2-visibility drawing. Since each iteration of the above induction takes $O(1)$ time, the lemma is proved.

LEMMA 5.2. *If H be an n-node plane triangulation, then H has an orderly spanning tree with at most $\frac{2}{3}n$ leaves that is obtainable in $O(n)$ time.*

Proof. (sketch) Chuang et al. [3] showed that three orderly spanning trees T_1, T_2, T_3 of H, with $H = T_1 \cup T_2 \cup T_3$, can be obtained in $O(n)$ time. For each $i = 1, 2, 3$, let $\text{leaf}(T_i)$ consist of the leaves of T_i. We show $\min_{i=1,2,3} |\text{leaf}(T_i)| \leq \frac{2}{3}n$. Let $v_{j_1}, v_{j_2}, \ldots, v_{j_{\ell_1}}$ be the leaves of T_1 where $j_1 < j_2 < \cdots < j_{\ell_1}$.

$$\begin{aligned} P_{123} &= \text{leaf}(T_1) \cap \text{leaf}(T_2) \cap \text{leaf}(T_3); \\ P_{12} &= (\text{leaf}(T_1) \cap \text{leaf}(T_2)) - \text{leaf}(T_3); \\ P_{13} &= (\text{leaf}(T_1) \cap \text{leaf}(T_3)) - \text{leaf}(T_2); \\ P_1 &= \text{leaf}(T_1) - (\text{leaf}(T_2) \cup \text{leaf}(T_3)). \end{aligned}$$

Clearly, each v_{j_i} belongs to exactly one of the above four disjoint sets. We first show $|P_{123}| \leq |P_1|$ by ensuring that for each $i = 1, 2, \ldots, \ell_1 - 2$, if $v_{j_i} \in P_{123} \cup P_{12}$, then $v_{j_{i+1}} \in P_{12} \cup P_1$. Since H is a plane triangulation and T_1 is a spanning tree of H, we know that v_{j_i} and $v_{j_{i+1}}$ are adjacent in H. One can easily verify that $(v_{j_i}, v_{j_{i+1}})$ is an edge in $T_2 \cup T_3$. Now, if v_{j_i} were in $P_{123} \cup P_{12}$, then v_{j_i} would belong to $\text{leaf}(T_2)$. Hence, $(v_{j_i}, v_{j_{i+1}})$ is an edge in T_3, implying that $v_{j_{i+1}}$ cannot be a leaf of T_3, and thus $v_{j_{i+1}} \in P_{12} \cup P_1$. Therefore, the lemma follows from

$$\begin{aligned} & |\text{leaf}(T_1)| + |\text{leaf}(T_2)| + |\text{leaf}(T_3)| \\ \leq\ & 3|P_{123}| + 2(|P_{12}| + |P_{13}|) + |P_1| + \\ & \quad 2(n - |P_{123} \cup P_{12} \cup P_{13} \cup P_1|) \\ \leq\ & 2|P_{123} \cup P_{12} \cup P_{13} \cup P_1| + \\ & \quad 2(n - |P_{123} \cup P_{12} \cup P_{13} \cup P_1|) \\ =\ & 2n. \end{aligned}$$

THEOREM 5.1. *An $n \times \frac{2}{3}n$ 2-visibility drawing of any n-node plane graph can be computed in $O(n)$ time.*

Proof. Let G be the input plane graph. Let H be obtained by triangulating G. By Lemmas 5.1 and 5.2, we know that an $n \times \frac{2}{3}n$ 2-visibility drawing of H and thus that of G can be obtained in $O(n)$ time.

6 Concluding remarks

Our orderly-pair algorithm appears to be a fundamental graph-algorithmic tool. Besides the applications shown in Sections 4 and 5, our algorithm gives an alternative way to compute a *realizer* [23] of a plane triangulation, which gives the best known straight-line drawing of planar graphs on the grids. Compared to the proof

given in [23], ours is quite simple: Given a plane triangulation G, let T be the orderly spanning tree of G computed by our algorithm. Let $v_1, v_2, \ldots, v_n$ be the counterclockwise preordering of T, where v_1, v_2, and v_n are the external nodes of G. Since each face of G is a triangle, one can easily verify that the B_2 and B_4 of each v_i, with $3 \leq i \leq n-1$, are not empty. Moreover, for each unrelated edge (v_i, v_j) with $3 \leq i < j \leq n-2$, we know that either the B_4 of v_i or the B_2 of v_j has more than one edge. Thus, we can determine the parent p_i (respectively, q_i) of v_i in T_p (respectively, T_q) as follows. If there is a v_i whose B_2 (respectively, B_4) has exactly one edge (v_i, v_j), then let $p_i = v_j$ (respectively, let $q_i = v_j$), and delete (v_i, v_j) from the B_2 (respectively, B_4) of v_i. This process iterates until p_i and q_i are defined for each $i = 3, 4, \ldots, n-1$. One can easily verify that this linear-time algorithm is well defined, and the resulting triple (T, T_p, T_q) is indeed a realizer of G.

Acknowledgment

We thank Xin He for helpful comments, We thank Ho-Lin Chen and Hsu-Chun Yen for bringing [8] to our attention. We thank Kai-Ju Liu for reading the paper carefully. We also thank Richie Chih-Nan Chuang and Yuan-Jiunn Wang for comments on early versions of this paper.

References

[1] T. C. Bell, J. G. Cleary, and I. H. Witten. *Text Compression*. Prentice-Hall, Englewood Cliffs, NJ, 1990.

[2] A. Brodnik and J. I. Munro. Membership in constant time and almost-minimum space. *SIAM J. Computing*, 28(5):1627–1640, 2000.

[3] R. C.-N. Chuang, A. Garg, X. He, M.-Y. Kao, and H.-I. Lu. Compact encodings of planar graphs via canonical ordering and multiple parentheses. In K. G. Larsen, S. Skyum, and G. Winskel, editors, *25th International Colloquium on Automata, Languages, and Programming*, LNCS 1443, pages 118–129, Aalborg, Denmark, 1998. Springer-Verlag.

[4] D. R. Clark. *Compact Pat Trees*. PhD thesis, University of Waterloo, 1996.

[5] R. F. Cohen, G. Di Battista, A. Kanevsky, and R. Tamassia. Reinventing the wheel: An optimal data structure for connectivity queries (extended abstract). In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 194–200, San Diego, California, 16–18 May 1993.

[6] H. De Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.

[7] P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, IT-21:194–203, 1975.

[8] U. Fößmeier, G. Kant, and M. Kaufmann. 2-visibility drawings of planar graphs. In G. DiBattista, editor, *5th International Symposium on Graph Drawing*, LNCS 1353, pages 155–168, Rome, Italy, 1997. Springer-Verlag.

[9] M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *J. Comput. Syst. Sci.*, 48(3):533–551, 1994.

[10] D. Harel and M. Sardas. An algorithm for straight-line drawing of planar graphs. *Algorithmica*, 20(2):119–135, 1998.

[11] X. He, M.-Y. Kao, and H.-I. Lu. Linear-time succinct encodings of planar graphs via canonical orderings. *SIAM J. Discrete Math*, pages 317–325, 1999.

[12] X. He, M.-Y. Kao, and H.-I. Lu. A fast general methodology for information-theoretically optimal encodings for graphs. *SIAM J. Computing*, 30(3):838–846, 2000.

[13] IEEE. *38th Annual Symposium on Foundations of Computer Science*, Miami Beach, Florida, 20–22 Oct. 1997.

[14] A. Itai and M. Rodeh. Representation of graphs. *Acta Informatica*, 17:215–219, 1982.

[15] G. Jacobson. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science*, pages 549–554, Research Triangle Park, North Carolina, 30 Oct.–1 Nov. 1989. IEEE.

[16] S. Kannan, M. Naor, and S. Rudich. Implicit representation of graphs. *SIAM J. Discrete Math*, 5:596–603, 1992.

[17] G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, 1996.

[18] G. Kant and X. He. Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. *Theoret. Comput. Sci.*, 172(1-2):175–193, 1997.

[19] K. Keeler and J. Westbrook. Short encodings of planar graphs and maps. *Discrete Applied Mathematics*, 58:239–252, 1995.

[20] J. I. Munro and V. Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *38th Annual Symposium on Foundations of Computer Science* [13], pages 118–126.

[21] M. Naor. Succinct representation of general unlabeled graphs. *Discrete Applied Mathematics*, 28:303–307, 1990.

[22] J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, 1999.

[23] W. Schnyder. Embedding planar graphs on the grid. In *1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 138–148, 1990.

[24] M. Talamo and P. Vocca. Compact implicit representation of graphs (extended abstract). In J. Hromkovic and O. Sýkora, editors, *Graph-theoretic concepts in computer science*, LNCS 1517, pages 164–176.

Springer, Smolenice Castle, Slovak Republic, 1998.
[25] M. Thorup. Undirected single source shortest paths in linear time. In *38th Annual Symposium on Foundations of Computer Science* [13], pages 12–21.
[26] M. Thorup. On RAM priority queues. *SIAM J. Computing*, 30:86–109, 2000.
[27] G. Turán. On the succinct representation of graphs. *Discrete Applied Mathematics*, 8:289–294, 1984.
[28] W. T. Tutte. A census of planar triangulations. *Canadian Journal of Mathematics*, 14:21–38, 1962.
[29] W. T. Tutte. A census of planar maps. *Canadian Journal of Mathematics*, 15:249–271, 1963.
[30] P. van Emde Boas. Machine models and simulations. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 1, pages 1–60. Elsevier, Amsterdam, 1990.
[31] M. Yannakakis. Embedding planar graphs in four pages. *J. Comput. Syst. Sci.*, 38(1):36–67, Feb. 1989.

A A linear-time implementation

In this appendix, we prove Theorem 2.1 by giving a linear-time implementation of opair. Suppose the input plane graph G is represented by an adjacency list, where each node v keeps a doubly linked list that stores its neighbors in counterclockwise order around v. Using this representation, both deleting an edge and moving an edge to the interior of a face can be done in $O(1)$ time. Two global arrays t and p are used in the algorithm. All the others are local variables. For each node v of G, $p[v]$ stores the parent of v in T; and $t[v]$ indicates whether v is external in the current G. Moreover, the algorithm uses $t[v] = 2$ to signify that v is the root of some 2-connected components of the current G. Initially, $\mathsf{opair}(G, v_1)$ makes a duplicated copy H of G. Let $t[v_1] = 2$. Let $t[v] = 1$ for each external node v of G with $v \neq v_1$. Let $t[v] = 0$ for all the other nodes of G. Then, $\mathsf{opair}(G, v_1)$ calls $\mathsf{block}(v_1, \mathrm{succ}(G, v_1))$. Finally, $\mathsf{opair}(G, v_1)$ outputs the resulting plane graph H and the tree T, which can be easily constructed from the resulting array p. During the execution of $\mathsf{opair}(G, v_1)$, the embeddings of G and H are iteratively changed, if necessary; while the nodes are deleted from G only. The process continues until all nodes are deleted from G. Let $\mathrm{ccw}(u, v)$ (respectively, $\mathrm{cw}(u, v)$) denote the neighbor of v that immediately succeeds u in counterclockwise (respectively, clockwise) order around v in the current G. For example, in Figure 3(b), we have $\mathrm{ccw}(1, 2) = 6$ and $\mathrm{cw}(2, 5) = 4$. The required subroutines are given in Figures 6. One can prove that the new implementation of the algorithm $\mathsf{opair}(G, v_1)$ correctly outputs an orderly pair (H, T) of G in linear time. The proof is omitted due to the page limit. Since the time complexity of our algorithm is dominated by that of opair, Theorem 2.1 follows immediately.

```
subroutine boundary(x, x', y) {
    while x ≠ y do {
        let x̂ = cw(x', x);
        let t[x] = t[x] + 1;
        if t[x] ≥ 2 then
            block(x, x');
        let t[x] = t[x] − 1;
        let (x, x') = (x̂, x);
    }
}

subroutine block(r, v) {
    let v' = r;
    mobile(v', v);
    while p[v] ≠ ccw(v', v) do
        block(v, ccw(v', v));
    let u = ccw(v', v);
    let u' = cw(v, u);
    remove v from G;
    if u ≠ v' then
        boundary(u', u, v');
}

subroutine flipin(v, x, w, x') {
    let x̂ = ccw(x', x);
    update G and H by moving the edge
        (v, x) to a face such that ccw(x', x)
        becomes v and cw(w, v) becomes x;
    boundary(x̂, x, v);
}

subroutine mobile(v', v) {
    let (w, p[v]) = (v', ⊥);
    while cw(w, v) ≠ ccw(v', v) do {
        let (x, x') = (w, v);
        while x ≠ v do
            if x ≠ ccw(v', v) then
                let (x, x') = (ccw(x', x), x);
            else flipin(v, x, w, x');
        if t[x'] ≥ 1 and p[v] = ⊥ then
            let p[v] = x';
        let w = cw(w, v);
    }
    if t[v'] = 2 then let p[v] = v';
    else if p[v] = ⊥ then let p[v] = ccw(v', v);
}
```

Figure 6: The subroutines for the linear-time implementation.