Compact Encodings of Planar Graphs via Canonical Orderings and Multiple Parentheses

Richie Chih-Nan Chuang[1], Ashim Garg[2], Xin He[2*], Ming-Yang Kao[3**], and Hsueh-I Lu[1]

[1] Department of Computer Science and Information Engineering, National Chung-Cheng University, Chia-Yi 621, Taiwan, {cjn85, hil}@cs.ccu.edu.tw
[2] Department of Computer Science, State University of New York at Buffalo, Buffalo, NY 14260, USA, {agarg,xinhe}@cs.buffalo.edu
[3] Department of Computer Science, Yale University, New Haven, CT 06250, USA, kao-ming-yang@cs.yale.edu

Abstract. We consider the problem of coding planar graphs by binary strings. Depending on whether $O(1)$-time queries for adjacency and degree are supported, we present three sets of coding schemes which all take linear time for encoding and decoding. The encoding lengths are significantly shorter than the previously known results in each case.

1 Introduction

This paper investigates the problem of encoding a graph G with n nodes and m edges into a binary string S. This problem has been extensively studied with three objectives: (1) minimizing the length of S, (2) minimizing the time needed to compute and decode S, and (3) supporting queries efficiently.

A number of coding schemes with different trade-offs have been proposed. The adjacency-list encoding of a graph is widely useful but requires $2m\lceil\log n\rceil$ bits. (All logarithms are of base 2.) A folklore scheme uses $2n$ bits to encode a rooted n-node tree into a string of n pairs of balanced parentheses. Since the total number of such trees is at least $\frac{1}{2(n-1)}\cdot\frac{(2n-2)!}{(n-1)!(n-1)!}$, the minimum number of bits needed to differentiate these trees is the log of this quantity, which is $2n-o(n)$. Thus, two bits per edge up to an additive $o(1)$ term is an information-theoretic tight bound for encoding rooted trees. Works on encodings of certain other graph families can be found in [7, 12, 4, 17, 5, 16].

Let G be a plane graph with n nodes, m edges, f faces, and no self-loop. G need not be connected or simple. We give coding schemes for G which all take $O(m+n)$ time for encoding and decoding. The bit counts of our schemes depend on the level of required query support and the structure of the encoded graphs.

For applications that require support of certain queries, Jacobson [6] gave an $\Theta(n)$-bit encoding for a simple planar graph G that supports traversal in $\Theta(\log n)$ time per node visited. Munro and Raman [15] recently gave schemes to encode a planar graph using $2m+8n+o(m+n)$ bits while supporting adjacency and degree queries in $O(1)$ time. We reduce this bit count to $2m+5\frac{1}{k}n+o(m+n)$ for any

* Research supported in part by NSF Grant CCR-9205982.
** Research supported in part by NSF Grant CCR-9531028.

	adjacency and degree		adjacency		no query	
	[15]	ours	old	ours	[13]	ours
self-loops					$3.58m$	
general	$2m+8n$	$2m+5\frac{1}{k}n$		$2m+4\frac{2}{3}n$		
simple		$\frac{5}{3}m+5\frac{1}{k}n$		$\frac{4}{3}m+5n$		
degree-one free					$3m$	
triconnected		$2m+3n$		$2m+3n$		
simple & triconnected		$2m+2n$		$2m+2n$		$\frac{3}{2}(\log 3)m$
triangulated		$2m+2n$		$2m+2n$		
simple & triangulated		$2m+n$		$2m+n$	$1.53m$	$\frac{4}{3}m$

Fig. 1. This table compares our results with previous ones, where k is a positive constant. The lower-order terms are omitted. All but row 1 assume that G has no self-loop.

constant $k > 0$ with the same query support. If G is triconnected or triangulated, our bit count decreases to $2m+3n+o(m+n)$ or $2m+2n+o(m+n)$, resp. With the same query support, we can encode a simple G using only $\frac{5}{3}m+5\frac{1}{k}n+o(n)$ bits for any constant $k > 0$. If a simple G is also triconnected or triangulated, the bit count is $2m+2n+o(n)$ or $2m+n+o(n)$, resp. If only $O(1)$-time adjacency queries are supported, our bit counts for a general G and a simple G become $2m+4\frac{2}{3}n+o(m+n)$ and $\frac{4}{3}m+5n+o(n)$, resp.

If we only need to reconstruct G with no query support, the code length can be substantially shortened. For this case, Turán [19] used $4m$ bits. This bound was improved by Keeler and Westbrook [13] to $3.58m$ bits. They also 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 G, we improve the count to $\frac{4}{3}m+O(1)$. For a simple G that is free of self-loops, triconnected and thus free of degree-one nodes, we improve the bit count to $1.5(\log 3)m+O(1)$. Figure 1 summarizes our results and compares them with previous ones.

Our coding schemes employ two new tools. One is new techniques of processing strings of multiple types of parentheses. The other tool is new properties of canonical orderings for plane graphs which were introduced in [3, 8]. These concepts have proven useful also for drawing plane graphs [10, 11, 18]. §2 discusses the new tools. §3 describes the coding schemes that support queries. §4 presents the more compact coding schemes which do not support queries. Due to space limitation, the proofs of most lemmas are omitted.

2 New Encoding Tools

A *simple* (resp., *multiple*) graph is one that does not contain (resp., may contain) multiple edges between two distinct vertices. A multiple graph can be viewed as a simple one with positive integral edge weights, where each edge's weight indicates its multiplicity. The *simple version* of a multiple graph is one obtained from the graph by deleting all but one copy of each edge. In this paper, all graphs are multiple unless explicitly stated otherwise. The *degree* of a node v in a graph

is the number of edges, counting multiple edges, incident to v in the graph. A node v is a *leaf* of a tree T if v has exactly one neighbor in T. Since T may have multiple edges, a leaf of T may have a degree greater than one.

2.1 Multiple Types of Parentheses

Let S be a string. S is *binary* if it contains at most two kinds of symbols. Let $S[i]$ be the symbol at the i-th position of S, for $1 \leq i \leq |S|$. 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 that precede or at the k-th position of S. Clearly, if $k = \mathrm{select}(S, i, \square)$, then $i = \mathrm{rank}(S, k, \square)$. Let $S_1 + \cdots + S_k$ denote the concatenation of strings $S_1, \ldots, S_k$. (In this paper, the encoding of G is usually a concatenation of several strings. For simplicity, we ignore the issue of separating these strings. This can be handled by using well-known data compression techniques with $\log n + O(\log \log n)$ bits [1].)

Let S be a string of multiple types of parentheses. Let $S[i]$ and $S[j]$ be an open and a close parenthesis with $i < j$ of the same type. $S[i]$ and $S[j]$ *match* in S if every parenthesis enclosed by $S[i]$ and $S[j]$ that is the same type as $S[i]$ and $S[j]$ matches a parenthesis enclosed by $S[i]$ and $S[j]$. Here are some queries defined for S:

- Let $\mathrm{match}(S, i)$ be the position of the parenthesis in S that matches $S[i]$.
- Let $\mathrm{first}_k(S, i)$ (resp., $\mathrm{last}_k(S, i)$) be the position of the first (resp., last) parenthesis of the k-th type that succeeds (resp., precedes) $S[i]$.
- Let $\mathrm{enclose}_k(S, i_1, i_2)$ be the positions (j_1, j_2) of the closest matching parenthesis pair of the k-th type that encloses $S[i_1]$ and $S[i_2]$.

S is *balanced* if every parenthesis in S belongs to a matching parenthesis pair. Note that the answer to a query above may be undefined. If there is only one type of parentheses in S, the subscript k in $\mathrm{first}_k(S, i)$, $\mathrm{last}_k(S, i)$, and $\mathrm{enclose}_k(S, i, j)$ may be omitted; thus, $\mathrm{first}(S, i) = i + 1$ and $\mathrm{last}(S, i) = i - 1$. If it is clear from the context, the parameter S may also be omitted.

Fact 1 ([2, 14, 15]) *1. Let S be a binary string. An auxiliary binary string $\mu_1(S)$ of length $o(|S|)$ can be obtained in $O(|S|)$ time such that* $\mathrm{rank}(S, i, \square)$ *and* $\mathrm{select}(S, i, \square)$ *can be answered from $S + \mu_1(S)$ in $O(1)$ time.*

2. Let S be a balanced string of one type of parentheses. An auxiliary binary string $\mu_2(S)$ of length $o(|S|)$ can be obtained in $O(|S|)$ time such that $\mathrm{match}(S, i)$ *and* $\mathrm{enclose}(S, i, j)$ *can be answered from $S + \mu_2(S)$ in $O(1)$ time.*

The next theorem generalizes Fact 1 to handle a string of multiple types of parentheses that is not necessarily balanced.

Theorem 1. *Let S be a string of $O(1)$ types of parentheses that may be unbalanced. An auxiliary $o(|S|)$-bit string $\alpha(S)$ can be obtained in $O(|S|)$ time such that* $\mathrm{rank}(S, i, \square)$, $\mathrm{select}(S, i, \square)$, $\mathrm{match}(S, i)$, $\mathrm{first}_k(S, i)$, $\mathrm{last}_k(S, i)$, *and* $\mathrm{enclose}_k(S, i, j)$ *can be answered from $S + \alpha(S)$ in $O(1)$ time.*

Proof. The statement for $\mathrm{rank}(S, i, \square)$ and $\mathrm{select}(S, i, \square)$ is a straightforward generalization of Fact 1(1). The statement for $\mathrm{first}_k(S, i)$ can be shown as follows. Let $f(S, i, \square)$ be the position of the first $\square$ that succeeds $S[i]$. Clearly,

$$f(S, i, \square) = \mathrm{select}(S, 1 + \mathrm{rank}(S, i, \square), \square); \;\; \mathrm{first}_k(S, i) = \min\{f(S, i, (), f(S, i,))\}$$

where (and) are the open and close parentheses of the k-th type in S, resp. The statement for $\text{last}_k(S, i)$ can be shown similarly.

To prove the statement for $\text{match}(S, i)$ and $\text{enclose}_k(S, i, j)$, first we ca show that Fact 1 can be generalized to an unbalanced binary string S (proof omitted). Suppose S has ℓ types of parentheses. Let S_k $(1 \leq k \leq \ell)$ be the string obtained from S as follows.

- Every open (resp., close) parenthesis of the k-th type is replaced by two consecutive open (resp., close) parentheses of the k-th type.
- Every parenthesis of any other type is replaced by a matching parenthesis pair of the k-th type.

Each S_k is a string of length $2|S|$ consisting of one type of parentheses and each symbol $S_k[i]$ can be determined from $S[\lfloor i/2 \rfloor]$ in $O(1)$ time. For example,

$$\begin{aligned} S &= \texttt{[[(\{)] (\{ \} \} ( ])} \\ S_1 &= \texttt{()()((()))()((()()()((()))}\\ S_2 &= \texttt{[[[[[][][]]][][][][][]]][]} \end{aligned}$$

The queries for S can be answered by answering the queries for S_k as follows.

- $\text{match}(S, i) = \lfloor \text{match}(S_k, 2i)/2 \rfloor$, where $S[i]$ is a parenthesis of the k-th type.
- Given i and j, let $A = \{2i, 2i+1, \text{match}(S_k, 2i), \text{match}(S_k, 2i+1)\} \cup \{2j, 2j+1, \text{match}(S_k, 2j), \text{match}(S_k, 2j+1)\}$. Let $i_1 = \min A$, $j_1 = \max A$, and $(i_2, j_2) = \text{enclose}(S_k, i_1, j_1)$. Then: $\text{enclose}_k(S, i, j) = (\lfloor i_2/2 \rfloor, \lfloor j_2/2 \rfloor)$.

Note that each of the above queries on some S_k can be answered in $O(1)$ time by $S_k + \mu_2(S_k)$. Since each symbol $S_k[i]$ can be determined from $S[\lfloor i/2 \rfloor]$ in $O(1)$ time, the theorem holds by letting $\alpha(S) = \mu_2(S_1) + \mu_2(S_2) + \cdots + \mu_2(S_\ell)$. □

Let $S_1, \ldots, S_k$ be k strings, each of $O(1)$ types of parentheses. For the remainder of the paper, let $\alpha(S_1, S_2, \ldots, S_k)$ denote $\alpha(S_1) + \alpha(S_2) + \cdots + \alpha(S_k)$.

2.2 Encoding Trees

An encoding for a graph G is *weakly convenient* if it takes linear time to reconstruct G; $O(1)$ time to determine the adjacency of two nodes in G; $O(d)$ time to determine the degree of a node; and $O(d)$ time to list the neighbors of a node of degree d. A weakly convenient encoding for G is *convenient* if it takes $O(1)$ time to determine the degree of a node.

The folklore encoding $F(T)$ of a simple rooted unlabeled tree T of n nodes uses a balanced string S of one type of parentheses to represent the preordering of T. Each node of T corresponds to a matching parenthesis pair in S.

Fact 2 *Let v_i be the i-th node in the preordering of a rooted simple tree T. The following properties hold for the folklore encoding S of T.*

1. *The parenthesis pair for v_i encloses the parenthesis pair for v_j in S if and only if v_i is an ancestor of v_j.*
2. *The parenthesis pair for v_i precedes the parenthesis pair for v_j in S if and only if v_i and v_j are not related and $i < j$.*

3. *The i-th open parenthesis in S belongs to the parenthesis pair for v_i.*

Fact 3 ([15]) *Let T be a simple rooted tree of n nodes. $F(T) + \mu_2(F(T))$ is a weakly convenient encoding for T of $2n + o(n)$ bits, obtainable in $O(n)$ time.*

We show Fact 3 holds even if S is mixed with other $O(1)$ types of parentheses.

Theorem 2. *Let T be a simple rooted unlabeled tree. Let S be a string of $O(1)$ types of parentheses such that a given type of parentheses in S gives the folklore encoding of T. Then $S + \alpha(S)$ is a weakly convenient encoding of T.*

Proof. Let the parentheses, denoted by (and), in S used by the encoding of T be the k-th type. Let $v_1, \ldots, v_n$ be the preordering of T. Let $p_i = \mathrm{select}(S, i, ()$ and $q_i = \mathrm{match}(S, p_i)$. By Theorem 1, p_i and q_i can be obtained from $S + \alpha(S)$ in $O(1)$ time. The index i can be obtained from p_i or q_i in $O(1)$ time by $i = \mathrm{rank}(S, p_i, ()) = \mathrm{rank}(S, \mathrm{match}(S, q_i), ())$. The queries for T are as follows.

Case: adjacency queries. Suppose $i < j$. Then, $(p_i, q_i) = \mathrm{enclose}_k(p_j, q_j)$ if and only if v_i is adjacent to v_j in T, i.e., v_i is the parent of v_j in T.

Case: neighbor queries. Suppose that v_i has degree d in T. The neighbors of v_i in T can be listed in $O(d)$ time as follows. First, if $i \neq 1$, output v_j, where $(p_j, q_j) = \mathrm{enclose}_k(p_i, q_i)$. Then, let $p_j = \mathrm{first}_k(p_i)$. As long as $p_j < q_i$, we repeatedly output v_j and update p_j by $\mathrm{first}_k(\mathrm{match}(p_j))$.

Case: degree queries. Since T is simple, the degree d of v_i in T is simply the number of neighbors in T, which is obtainable in $O(d)$ time. □

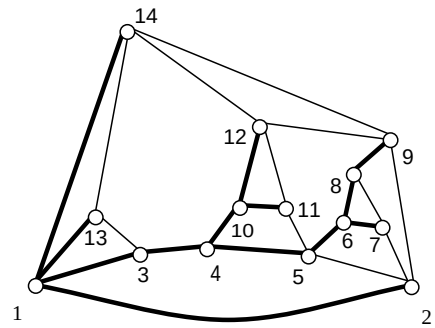
We next improve Theorem 2 to obtain convenient encodings for multiple trees. For a condition P, let $\delta(P) = 1$, if P holds; let $\delta(P) = 0$, otherwise.

Theorem 3. *Let T be a rooted unlabeled tree of n nodes, n_1 leaves and m edges. Let $S + \alpha(S)$ be a weakly convenient encoding of T_s (the simple version of T).*

1. *A string D of $(2m - n + n_1)$ bits can be obtained in $O(m + n)$ time such that $S + D + \alpha(S, D)$ is a convenient encoding for T of $2m + n + n_1 + o(m)$ bits.*
2. *If T is simple, a string D of n_1 bits and a string Y of n bits can be obtained in $O(m + n)$ time such that $S + D + \alpha(S, D, Y)$ is a convenient encoding for T and has $2n + n_1 + o(n)$ bits.*

Proof. Let $v_1, \ldots, v_n$ be the preordering of T_s. Let d_i be the degree of v_i in T. We show how to use a string D to store the information required to obtain d_i in $O(1)$ time. We only prove Statement 1.

Let $\delta_i = \delta(v_i \text{ is internal in } T_s)$. Since $S + \alpha(S)$ is a weakly convenient encoding for T_s, each δ_i can be obtained in $O(1)$ time from $S + \alpha(S)$. Initially, D is just n copies of 1. Let $b_i = d_i - 1 - \delta_i$. We add b_i copies of 0 right after the i-th 1 in D for each v_i. Since the number of internal nodes in T_s is $n - n_1$, the bit count of D is $n + \sum_{i=1}^{n}(d_i - 1 - \delta_i) = n + 2m - n - (n - n_1) = 2m - n + n_1$. D can be obtained from T in $O(m + n)$ time. The number b_i of 0's right after the i-th 1 in D is $\mathrm{select}(D, i + 1, 1) - \mathrm{select}(D, i, 1) - 1$. Since $d_i = 1 + \delta_i + b_i$, the degree of v_i in T can be computed in $O(1)$ time from $S + D + \alpha(S, D)$. □



step j :	interval I_j :
1	3, 4, 5
2	6, 7
3	8
4	9
5	10, 11
6	12
7	13
8	14

Fig. 2. A triconnected plane graph G and a canonical ordering of G.

2.3 Canonical Orderings

In this subsection, we describe the *canonical ordering* of plane graphs. It was first introduced for plane triangulations in [3], and extended to triconnected plane graphs in [8]. We prove some new properties of this ordering. Let G be a simple triconnected plane graph. Let $v_1, \ldots, v_n$ be a node ordering of G. Let G_i be the subgraph of G induced by $v_1, v_2, \ldots, v_i$. Let H_i be the exterior face of G_i.

Definition 1. Let $v_1, v_2, \ldots, v_n$ be a node ordering of a simple triconnected plane graph $G = (V, E)$, where (v_1, v_2) is an arbitrary edge on the exterior face of G. The ordering is *canonical* if there exist ordered intervals $I_1, \ldots, I_K$ that partition the interval $[3, n]$ such that the following properties hold for every $1 \leq j \leq K$: Suppose $I_j = [k, k+q]$. Let C_j be the path $(v_k, v_{k+1}, \ldots, v_{k+q})$.

- The graph G_{k+q} is biconnected. Its boundary H_{k+q} contains the edge (v_1, v_2) and the path C_j. C_j has no chords in G.
- If $q = 0$, v_k has at least two neighbors in G_{k-1}, each of them is on H_{k-1}.
- If $q > 0$, the path C_j has exactly two neighbors in G_{k-1}, each of them is on H_{k-1}. The leftmost neighbor v_ℓ is incident only to v_k and the rightmost neighbor v_r is incident only to v_{k+q}.
- For each v_i $(k \leq i \leq k+q)$, if $i < n$, v_i has at least one neighbor in $G - G_{k+q}$.

Figure 2 shows a canonical ordering of G. Every triconnected plane graph has a canonical ordering which can be constructed in $O(n)$ time [8].

Given a canonical ordering of G with interval partition $I_1, I_2, \ldots, I_K$, we can obtain $G = G_n$ from G_2, which consists of the single edge (v_1, v_2), through the following K steps: Suppose $I_j = [k, k+q]$. The j-th step obtains G_{k+q} from G_{k-1} by adding $q+1$ nodes $v_k, v_{k+1}, \ldots, v_{k+q}$ and their incidental edges in G_{k+q}.

Let T be the edge (v_1, v_2) plus the union of the paths $(v_\ell, v_k, v_{k+1}, \ldots, v_{k+q})$ over all intervals $I_j = [v_k, v_{k+q}]$, $1 \leq j \leq K$, where v_ℓ is the leftmost neighbor of v_k on H_{k-1}. One can easily see that T is a spanning tree of G rooted at v_1. T is called a *canonical spanning tree* of G. In Figure 2, T is indicated by thick lines.

We show every canonical spanning tree T has the following property.

Lemma 1. *Let T be the canonical spanning tree rooted at v_1 corresponding to a canonical ordering $v_1, v_2, \ldots, v_n$ of G.*

1. *Let $(v_i, v_{i'})$ be an edge in $G - T$. Then v_i and $v_{i'}$ are not related in T.*
2. *For each node v_i, the edges incident to v_i show the following pattern around v_i in counterclockwise order: The edge from v_i to its parent in T; followed by a block of nontree edges from v_i to lower-numbered nodes; followed by a block of tree edges from v_i to its children in T; followed by a block of nontree edges from v_i to higher-numbered nodes. (Any of these blocks maybe empty).*

3 Schemes with Query Support

In this section we present our coding schemes that support queries. We give a weakly convenient encoding for a simple triconnected graph G in §3.1, which illustrates our basic techniques. We give the schemes for triconnected plane graphs in §3.2. We state our results for triangulated and general plane graphs in §3.3.

3.1 Basis

Let T be a canonical spanning tree of a simple triconnected plane graph G. We encode G using a balanced string S of two types of parentheses. The first type (*parentheses*) is for the edges of T. The second type (*brackets*) is for the edges of $G - T$.

The encoding Let S be the folklore encoding for T. Let v_i be the i-th node in the counterclockwise preordering of nodes of T. Let $(_i$ and $)_i$ be the parenthesis pair corresponding to v_i in S. We augment S by inserting a pair $[_e$ and $]_e$ of brackets for every edge $e = (v_i, v_j)$, where $i < j$, of $G - T$ as follows: we place $[_e$ right after $)_i$ and $]_e$ right after $(_j$.

Suppose that v_i is adjacent to ℓ_i (resp., h_i) lower- (higher-, resp.) numbered nodes in $G - T$. Then S has the following pattern for every $1 \leq i \leq n$: The open parenthesis $(_i$ is immediately followed by ℓ_i close brackets. The close parenthesis $)_i$ is immediately followed by h_i open brackets. The following properties are clear.

Fact 4 *Let $e = (v_i, v_j)$ be an edge of $G - T$, where $i < j$. Then*

1. *$[_e$ is located between $)_i$ and the first parenthesis that succeeds $)_i$ in S;*
2. *$]_e$ is located between $(_j$ and the first parenthesis that succeeds $(_j$ in S.*

The following property for S is immediate from Fact 4:
Property A: The last parenthesis that precedes an open bracket is close. The last parenthesis that precedes a close bracket is open.

Let $e = (v_i, v_j)$ be an edge of $G - T$, where $i < j$. By Lemma 1 and Fact 2, $)_i$ precedes $(_j$ in S. By Fact 4, S has the following property:

Fact 5 *Let e be an edge of $G - T$. Then $[_e$ precedes $]_e$ in S.*

Lemma 2. *Let e and f be two edges in $G - T$ with no common end vertex. Suppose that $[_e < [_f$. Then either $[_e <]_e < [_f <]_f$ or $[_e < [_f <]_f <]_e$.*

($[_e < [_f$ indicates $[_e$ precedes $[_f$.) The above lemma implies that $]_e$ and the bracket that matches $[_e$ in S are in the same block of brackets. From now on, we rename the close brackets by redefining $]_e$ to be the close bracket that matches $[_e$ in S. It is clear that Property A and Facts 4, 5 still hold for S.

The queries We show $S + \alpha(S)$ is a weakly convenient encoding for G. Since T is simple, then by Theorem 2, $S+\alpha(S)$ is a weakly convenient encoding for T. It remains to show that $S + \alpha(S)$ is also a weakly convenient encoding for $G - T$. Let p_i and q_i be the positions of $(_i$ and $)_i$ in S, resp.

- *Adjacency.* Suppose $i < j$. Note that v_i and v_j are adjacent in $G - T$ if and only if $q_i < p < q < \text{first}_1(p_j)$, where $(p, q) = \text{enclose}_2(\text{first}_1(q_i), p_j)$, as indicated by the following figure:

$$\begin{array}{cccccccc})_i & [& & & (_j &] & & \\ \uparrow & \uparrow & & \uparrow & \uparrow & \uparrow & & \uparrow \\ q_i & p & \text{first}_1(q_i) & & p_j & q & \text{first}_1(p_j) & \end{array}$$

- *Neighbors and degree.* The neighbors, and thus the degree, of a degree-d node v_i in $G - T$ can be obtained in $O(d)$ time as follows.
  - For every position p such that $q_i < p < \text{first}_1(q_i)$, we output v_j, where $p_j = \text{last}_1(\text{match}(p))$. ($(v_i, v_j)$ is an edge in $G - T$ with $j > i$.)
  - For every position q such that $p_i < q < \text{first}_1(p_i)$, we output v_j, where $q_j = \text{last}_1(\text{match}(q))$. ($(v_i, v_j)$ is an edge in $G - T$ with $j < i$.)

The bit count. Clearly $|S| = 2n+2(m-n) = 2m$. Since there are four symbols in S, S can be encoded by $4m$ bits. We can improve the bit count by the following:

Lemma 3. *Let S be a string of p parentheses and b brackets that satisfies Property A. Then S can be encoded by a string of $2p + b + o(p + b)$ bits, from which each $S[i]$ can be determined in $O(1)$ time.*

Proof. Let S_1 and S_2 be two binary strings defined as follows.

- $S_1[i] = 1$ if and only if $S[i]$ is a parenthesis, $1 \le i \le p + b$.
- $S_2[j] = 1$ if and only if the j-th parenthesis in S is open, $1 \le j \le p$.

Each $S[i]$ can be determined from $S_1 + S_2 + \alpha(S_1)$ in $O(1)$ time as follows. Let $j = \text{rank}(S_1, i, 1)$. If $S_1[i] = 1$, $S[i]$ is a parenthesis. Whether it is open or close can be determined from $S_2[j]$. If $S_1[i] = 0$, $S[i]$ is a bracket. Whether it is open or close can be determined from $S_2[\text{select}(S_1, \text{rank}(S_1, i, 1), 1)]$ by Property A. □

We summarize the above arguments as follows.

Lemma 4. *A simple triconnected plane graph of n nodes and m edges has a weakly convenient encoding that has $2m + 2n + o(n)$ bits.*

3.2 Triconnected Plane Graphs

We adapt all notation of §3.1 to this subsection. We first show that the weakly convenient encoding for a simple triconnected plane graph G given in §3.1 can be further shortened to $2(m + n - n_1) + o(n)$, where n_1 is the number of leaves in T. We then give a convenient encoding for G that has $2m + 2n + o(n)$ bits. Finally we augment both encodings to handle multiple edges.

Let v_i be a leaf of T, where $2 < i < n$. By definition of T and Definition 1, v_i is adjacent to a higher-numbered node and a lower-numbered node in $G - T$. This implies that $(_i$ is immediately succeeded by a], and $)_i$ is immediately succeeded by a [, for every such v_i. Let P be the string obtained from S by removing a] that immediately succeeds $(_i$, and removing a [that immediately succeeds $)_i$ for every leaf v_i of T, where $2 < i < n$. If each $S[j]$ were obtainable in $O(1)$ time from $P + \alpha(P)$, the string S could then be replaced by $P + \alpha(P)$. This does not seem likely. However, we can show that there exists a string Q of length $|P|$, each $Q[i]$ can be obtained from $P + \alpha(P)$ in $O(1)$ time, such that $P + \alpha(P, Q)$ is a weakly convenient encoding for G. Since S satisfies Property A and P is obtained from S by removing some brackets, P also satisfies Property A. Since P has $2n$ parentheses and $2(m - (n-1) - n_1)$ brackets, by Lemma 3 G has a weakly convenient encoding of $2(m + n - n_1) + o(n)$ bits.

Next we augment our weakly convenient encoding for G to a convenient one. Note that the degree of v_i in $G - T$ can be obtained in $O(1)$ time from $P + \alpha(P, Q)$. It remains to supply $O(1)$-time degree query for T. By Theorem 3 we know that $n_1 + o(n)$ more bits suffices. Therefore there exists a $(2m + 2n - n_1 + o(n))$-bit convenient encoding for G that can be obtained in $O(m + n)$ time.

The above convenient encoding can be extended to handle multiple edges as follows. Let G_a be a multiple graph obtained from G by adding some multiple edges between nodes that are adjacent in $G - T$. Note that the above arguments in this subsection also hold for G_a exactly the same way. Suppose that G_a has m_a edges. Then G_a has a weakly convenient encoding of $2(m_a + n - n_1) + o(m_a + n)$ bits, from which the degree of a node in $G_a - T$ can actually be determined in $O(1)$ time. Let G_b be a multiple graph obtained from G_a by adding some multiple edges between nodes that are adjacent in T. Suppose that G_b has m_b edges. Let T_b be the union of multiple edges of G_b between the nodes that are adjacent in T. In order to obtain a convenient encoding for G_b, it remains to supply $O(1)$-time query for the degree of a node in T_b. Clearly T_b has $m_b - m_a + n - 1$ edges. By Theorem 3, $2(m_b - m_a + n - 1) - n + n_1 + o(m_b)$ more bits suffice.

We summarize the subsection as follows.

Lemma 5. *Let G be a triconnected plane graph of n nodes and m edges. Let G_s be the simple version of G, which has m_s edges. Let n_1 be the number of leaves in a canonical spanning tree of G_s. Then G (resp., G_s) has a convenient encoding of $2m + 3n - n_1 + o(m + n)$ (resp., $2m_s + 2n - n_1 + o(n)$) bits. All these encodings can be obtained in linear time.*

3.3 Plane Triangulations and General Plane Graphs

Lemma 6. *Let G be a plane triangulation of $n \geq 3$ nodes and m edges. Let G_s be the simple version of G, which has $m_s = 3n - 6$ edges. Then G (resp., G_s) has a convenient encoding of $2m + 2n + o(m + n)$ (resp., $2m_s + n + o(n)$) bits. All these encodings can be obtained in linear time.*

Lemma 7. *Let G be a plane graph of n nodes and m edges. Let G_s be the simple version of G, which has m_s edges. Let k be a positive constant. Then G has a convenient encoding of $2m + 5\frac{1}{k}n + o(m+n)$ bits and a weakly convenient encoding of $2m + 4\frac{2}{3}n + o(m + n)$ bits. G_s has a convenient encoding of $\frac{5}{3}m_s + 5\frac{1}{k}n + o(n)$ bits and a weakly convenient encoding of $\frac{4}{3}m_s + 5n + o(n)$ bits.*

4 More Compact Schemes

In some applications, the only requirement for the encoding is to reconstruct the graph, no queries are needed. In this case, we can obtain even more compact encodings for simple triconnected and triangulated plane graphs.

Let G be a simple triconnected plane graph. Let T be a canonical spanning tree of G. Let $v_1, \ldots, v_n$ be the counterclockwise preordering of T. By using techniques in [8], it can be shown that this ordering is also a canonical ordering of G. (In Figure 2, the canonical ordering shown is the counterclockwise preordering of T.) This special canonical ordering is used in our encoding.

Let $I_1, \ldots, I_K$ be the interval partition corresponding to the canonical ordering. G can be constructed from a single edge (v_1, v_2) through K steps. The j-th step corresponds to the interval $I_j = [k, k+q]$. There are two cases:

Case 1: A single node v_k is added.

Case 2: A chain of $q+1$ $(q > 0)$ nodes $v_k, \ldots, v_{k+q}$ is added.

The last node added during a step is called a *type a* node. Other nodes are *type b* nodes. Thus the single node v_k added during a Case 1 step is of type a. For a Case 2 step, the nodes $v_k, \ldots, v_{k+q-1}$ are of type b and v_{k+q} is of type a.

Consider the interval $I_j = [k, k+q]$. Let $c_1(= v_1), c_2, \ldots, c_t(= v_2)$ be the nodes of the exterior face H_{k-1} ordered consecutively along H_{k-1} from left to right above the edge (v_1, v_2). We define the following terms.

Case 1. Let c_ℓ and c_r $(1 \le \ell < r \le t)$ be the leftmost and rightmost neighbors of v_k in H_{k-1}, resp. The edge (c_ℓ, v_k) is in T. The edge (c_r, v_k) is called an *external edge*. The edges (c_i, v_k) where $\ell < i < r$, if present, are *internal edges*.

Case 2. Let c_ℓ and c_r $(1 \le \ell < r \le t)$ be the neighbors of v_k and v_{k+q} in H_{k-1}, resp. The edges $(c_\ell, v_k), (v_k, v_{k+1}), \ldots, (v_{k+q-1}, v_{k+q})$ are in T. The edge (c_r, v_k) is called an external edge.

For each v_k $(1 \le k \le n-1)$, let $B(v_k)$ denote the edge set $\{(v_k, v_j) \mid k < j\}$. By Definition 1 and Lemma 1, the edges in $B(v_k)$ show the following pattern around v_k in counterclockwise order: A block (maybe empty) of tree edges; followed by at most one internal edge; followed by a block (maybe empty) of external edges. Next, we show that if we know the sets $B(v_k)$ $(1 \le k \le n-1)$ and the type of v_k $(3 \le k \le n)$, then we can uniquely reconstruct G.

First the edge (v_1, v_2) is drawn. Then we perform the following K steps. The j-th step processes $I_j = [k, k+q]$. Before the j-th step, the graph G_{k-1} and its exterior face H_{k-1} has been constructed. We need to determine the leftmost neighbor c_ℓ and the rightmost neighbor c_r of the nodes added in this step. We know (c_ℓ, v_k) is a tree edge in T. Since $v_1, \ldots, v_n$ is the counterclockwise preordering of T, c_ℓ is the rightmost node that has a remaining tree edge and c_r is the leftmost node that is to the right of c_ℓ and has a remaining external edge. There are two cases:

If v_k is of type a, this is a Case 1 step and v_k is the single node added during this step. We add the edges (c_ℓ, v_k) and (c_r, v_k). For each c_i with $\ell < i < r$, if $B(c_i)$ contains an internal edge, we also add the edge (c_i, v_k).

If v_k is of type b, this is a Case 2 step. Let q be the integer such that $v_k, v_{k+1}, \ldots, v_{k+q-1}$ are of type b and v_{k+q} is of type a. The chain $v_k, \ldots, v_{k+q}$ is added between c_ℓ and c_r.

This completes the j-th step. When the process terminates, we obtain the graph G. Thus, if we can encode the type of each v_k and the sets $B(v_k)$ $1 \le$

$k \leq n-1$, then we get an encoding of G. We first define the *type* of a set $B(v_k)$, which tells us the types of the edges contained in $B(v_k)$. We use T to denote the tree edges, X the external edges, and I the internal edges. The type of $B(v_k)$ is a combination of the symbols T, X, I. For examples, if $B(v_k)$ has type TXI, then $B(v_k)$ contains tree edges, external edges and an internal edge, and so on.

We further divide type a nodes v_k into two subtypes: If $B(v_k)$ contains no tree edges, then v_k is a *type a1* node. If $B(v_k)$ contains tree edges, then v_k is a *type a2* node. For a type b node v_k, since v_k is not the last node added during a Case 2 step, by the definition of T, $B(v_k)$ contains at least one tree edge.

Our encoding of G uses two strings S_1 and S_2 both using three symbols $0, 1, *$. The length of S_1 is n. $S_1[k]$ $(1 \leq k \leq n)$ indicates whether v_k is of type a1, a2, or b. S_2 encodes the sets $B(v_k)$ $(1 \leq k \leq n-1)$. Each $B(v_k)$ is specified by a *code word*, denoted by Code$[v_k]$. S_2 is the concatenation of Code$[v_k]$ $(1 \leq k \leq n-1)$. The length of Code$[v_k]$ equals to the number of the edges in $B(v_k)$. Depending on the type of v_k and the type of $B(v_k)$, Figure 3 gives the format of Code$[v_k]$. In the table, the number of the tree edges (external edges, resp.) in $B(v_k)$ is denoted by α (β, resp). 1^α denotes a string of α copies of 1, and so on. A symbol T (resp., X or I) under Code$[v_k]$ denotes the portion in Code$[v_k]$ corresponding to the tree (resp., external or internal) edges.

Type of v_k	Type of $B(v_k)$	Code$[v_k]$	Type of v_k	Type of $B(v_k)$	Code$[v_k]$
a1	XI	$\underbrace{1^\beta}_{X}\underbrace{0}_{I}$	a2 or b	T	$\underbrace{0^{\alpha-1}*}_{T}$
	I	$\underbrace{0}_{I}$		TXI	$\underbrace{1^\alpha}_{T}\underbrace{0^\beta}_{X}\underbrace{*}_{I}$
	X	$\underbrace{1^{\beta-1}*}_{X}$		TX	$\underbrace{1^{\alpha-1}0}_{T}\underbrace{0^{\beta-1}1}_{X}$
				TI	$\underbrace{1^\alpha}_{T}\underbrace{*}_{I}$

Fig. 3. Code Word Table.

From S_1, S_2 and the Code Word Table, we can easily recover the type of each v_k and the sets $B(v_k)$. It is straightforward to implement the encoding and decoding procedures in $O(n)$ time. The length of S_1 is n. The length of S_2 is m. We use the binary representation S of S_1 and S_2 to encode G. Since both S_1 and S_2 use 3 symbols, $|S| = \log 3(n+m)$. Thus we have the following:

Lemma 8. *Any simple triconnected plane graph with n nodes and m edges can be encoded using at most* $\log 3(n+m)$ *bits. Both encoding and decoding procedures take $O(n)$ time.*

We can improve Lemma 8 as follows. Let G^* be the dual of G. G^* has f nodes, m edges and n faces. Since G is triconnected, so is G^*. Furthermore, if $n > 3$, then $f > 3$ and G^* has no self-loop or multiple edge. Thus, we can use the coding scheme of Lemma 8 to encode G^* with at most $\log 3(f+m)$ bits. Since G can be uniquely determined from G^*, to encode G, it suffices to encode

G^*. To make S shorter, if $n \leq f$, we encode G using at most $\log 3(n+m)$ bits; otherwise, we encode G^* using at most $\log 3(f+m)$ bits. This new encoding uses at most $\log 3(\min\{n,f\}+m)$ bits. Since $\min\{n,f\} \leq \frac{n+f}{2}$, the bit count is at most $\log 3(1.5m+1)$ by Euler's formula $n+f=m+2$. We use one extra bit to denote whether we encode G or G^*. Thus we have proved the following:

Theorem 4. *Any simple triconnected plane graph with n nodes, m edges and f faces can be encoded using at most* $\log 3(\min\{n,f\}+m)+1 \leq 1.5(\log 3)m+3$ *bits. Both encoding and decoding take $O(n)$ time.*

Theorem 5. *Any simple plane triangulation of n nodes and m edges can be encoded using $4n-7=\frac{4m}{3}+1$ bits. Both encoding and decoding take $O(n)$ time.*

References

1. T. Bell, J. G. Cleary, and I. Witten, *Text Compression*, Prentice-Hall, 1990.
2. D. R. Clark, *Compact Pat Tree*, PhD thesis, University of Waterloo, 1996.
3. H. D. Fraysseix, J. Pach, and R. Pollack, *How to draw a planar graph on a grid*, Combinatorica, 10 (1990), pp. 41–51.
4. H. Galperin and A. Wigderson, *Succinct representations of graphs*, Information and Control, 56 (1983), pp. 183–198.
5. A. Itai and M. Rodeh, *Representation of graphs*, Acta Informatica, 17 (1982), pp. 215–219.
6. G. Jacobson, *Space-efficient static trees and graphs*, in proc. 30th FOCS, 30 Oct.–1 Nov. 1989, pp. 549–554.
7. S. Kannan, N. Naor, and S. Rudich, *Implicit representation of graphs*, SIAM Journal on Discrete Mathematics, 5 (1992), pp. 596–603.
8. G. Kant, *Drawing planar graphs using the lmc-ordering (extended abstract)*, in proc. 33rd FOCS, 24–27 Oct. 1992, pp. 101–110.
9. ———, *Algorithms for Drawing Planar Graphs*, PhD thesis, Univ. of Utrecht, 1993.
10. G. Kant and X. He, *Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems*, TCS 172 (1997), pp. 175–193.
11. M. Y. Kao, M. Fürer, X. He, and B. Raghavachari, *Optimal parallel algorithms for straight-line grid embeddings of planar graphs*, SIAM Journal on Discrete Mathematics, 7 (1994), pp. 632–646.
12. M. Y. Kao and S. H. Teng, *Simple and efficient compression schemes for dense and complement graphs*, in Fifth Annual Symposium on Algorithms and Computation, LNCS 834, Beijing, China, 1994, Springer-Verlag, pp. 201–210.
13. K. Keeler and J. Westbrook, *Short encodings of planar graphs and maps*, Discrete Applied Mathematics, 58 (1995), pp. 239–252.
14. J. I. Munro, *Tables*, in proc. of 16th Conf. on Foundations of Software Technology and Theoret. Comp. Sci., LNCS 1180, 1996, Springer-Verlag, pp. 37–42.
15. J. I. Munro and V. Raman, *Succinct representation of balanced parentheses, static trees and planar graphs*, in proc. 38th FOCS 20–22 Oct. 1997.
16. M. Naor, *Succinct representation of general unlabeled graphs*, Discrete Applied Mathematics, 28 (1990), pp. 303–307.
17. C. H. Papadimitriou and M. Yannakakis, *A note on succinct representations of graphs*, Information and Control, 71 (1986), pp. 181–185.
18. W. Schnyder, *Embedding planar graphs on the grid*, in Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, 1990, pp. 138–148.
19. G. Turán, *On the succinct representation of graphs*, Discrete Applied Mathematics, 8 (1984), pp. 289–294.