クエリを効率的にサポートする極大平面グラフのコンパクトな符号化

山中 克久　　中野 眞一
群馬大学工学部情報工学科

概要　本文は, 各極大平面グラフを $2m + o(n)$ bit に符号化する簡単な方法を与える. ここで, m はグラフの辺数であり, n はグラフの点数である. この $2m + o(n)$ bit の符号から, 次の 3 つのクエリの解を定数時間で計算できる. 指定した 2 点が隣接するかどうか, 指定した 1 点の次数, 指定した 1 点とその隣接点が与えられたとき時計回りに次に隣接する点である. 本文の手法は, 極大平面グラフのリアライザに基づいている. これまで, 各極大平面グラフを $2m + n + o(n)$ bits に符号化する手法しか知られていなかった.

Compact Encoding of Plane Triangulations with Efficient Query Support

Katsuhisa Yamanaka and Shin-ichi Nakano
Gunma University, Kiryu-Shi 376-8515, Japan.

Abstract　In this paper we give a simple coding scheme for plane triangulations. The coding scheme needs $2m + o(n)$ bits for each plane triangulation, and supports adjacency, degree and clockwise neighbour queries in constant time. Our scheme is based on a realizer of a plane triangulation. The best known scheme needs $2m + (5 + 1/k)n + o(m + n)$ bits for each (general) plane graphs, and $2m + n + o(n)$ bits for each plane triangulation.

1　Introduction

Given a class C of graphs how many bits are needed to encode a graph $G \in C$ into a binary string S_G so that S_G can be decoded to reconstruct G. If C contains n_C graphs, then for any coding scheme the average length of S_G is at least $\log n_C$ bits, which is called *the information-theoretically optimal bound.*

By using any generating algorithm, we can encode the k-th generated graph into the binary representation of k, and attain the optimal bound. However such method may need exponential time for encoding and decoding.

On the other hand, for many application, efficient running time for encoding and decoding is required. Thus for various classes of graphs many coding schemes with efficient running time have been proposed. Moreover, some of those coding schemes support several graph operations in constant time. See [CG98, C01, C98, H99, H00, J89, KW95, MR97, MR01, T84].

In this paper we consider the problem for plane triangulations. We wish to design a scheme to encode a given plane triangulation G into a binary string S_G so that (1) S_G can be efficiently decoded to reconstruct G, (2) the length of S_G is short, and (3) S_G supports several graph operations in constant time.

The following results are known for the problem. Let m be the number of edges in a graph.

[T62] shows that the information-theoretically optimal bound is $1.08m$ bits for plane triangulations. However this coding scheme needs exponential time for encoding.

For schemes without any query support the following results are known. [T84] gives a scheme to encode a general planar graph into asymptotically $4m$ bits. [KW95] gives schemes to encode a general planar graph into $m \log 12 = 3.58m$ bits, a triconnected planar graph into $3m$ bits, and a plane triangulation into $(3 + \log 3)m/3 = 1.53m$ bits. [H99] gives a scheme based on "the canonical ordering" to encode a plane triangulation into $4m/3 - 1$ bits. [P03] gives a scheme based on a bijection with a class of trees to encode a plane triangulation into $4m/3$ bits.

For schemes with query support the following results are known. [J89] gives a scheme to encode trees achieving the information-theoretically optimal bound to within a lower order term, and still supporting some natural query operations quickly. [MR97, MR01] gives a scheme to encode a planar graph into $2m + 8n + o(n)$ bits with supporting adjacency and degree query in constant time. [CG98] gives a scheme to encode a planar graph into $2m + (5 + 1/k)n + o(n)$ bits, where k is any constant, and a plane triangulation into $2m + n + o(n)$ bits. [C01] gives a scheme to encode a planar graph into $2m + 2n + o(n)$ bits.

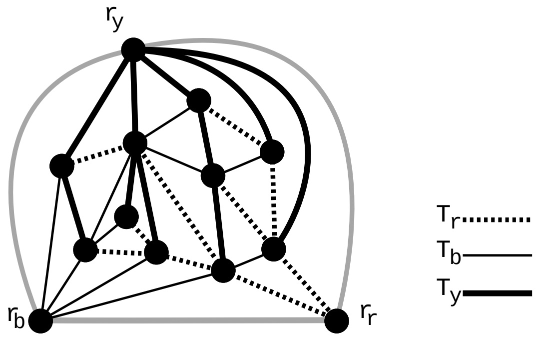

Figure 1: An example of a realizer.

In this paper we improve the best known result [CG98] for plane triangulations.

The class of plane triangulations is an important class of graphs, since the standard representation for 3D models, called triangle meshes, consists of vertex data and connectivity data[R99]. If the triangle mesh is homeomorphic to a sphere then the connectivity data is a plane triangulation.

We give a simple coding scheme for plane triangulations. The coding scheme needs only $2m + o(n)$ bits for each plane triangulation, and still supports adjacency and degree queries in $O(1)$ time. Given a vertex u and its neighbour v, many plane graph algorithms need to find the "next" neighbour of u succeeding v in clockwise order, because with this query one can trace a face, and it is one of basic operation for plane graph algorithms. Our coding scheme also find such a neighbour in $O(1)$ time. Our algorithm is based on a realizer[S90] (See an example in Fig. 1.) of a plane triangulation.

The rest of the paper is organized as follows. Section 2 gives some definitions. Section 3 introduces a realizer of a plane triangulation. Section 4 presents our coding scheme. In Section 5 we explain query support. Finally Section 6 is a conclusion.

2 Preliminaries

In this section we give some definitions.

Let $G = (V, E)$ be a connected graph with vertex set V and edge set E. We denote $n = |V|$ and $m = |E|$. An edge connecting vertices x and y is denoted by (x, y). The *degree* of a vertex v, denoted by $d(v)$, is the number of neighbours of v in G.

A graph is *planar* if it can be embedded in the plane so that no two edges intersect geometrically except at a vertex to which they are both incident. A *plane* graph is a planar graph with a fixed planar embedding. A plane graph divides the plane into connected regions called *faces.* The unbounded face is called *the outer face*, and other faces are called *inner faces.* We regard *the contour* of a face as the clockwise cycle formed by the vertices and edges on the boundary of the face. We denote the contour of the outer face of plane graph G by $C_o(G)$. A vertex is *an outer vertex* if it is on $C_o(G)$, and *an inner vertex* otherwise. An edge is *an outer edge* if it is on $C_o(G)$, and *an inner edge* otherwise. A plane graph is called *a plane triangulation* if each face has exactly three edges on its contour. By Euler's Formula: $n - m + f = 2$, where f is the number of faces, one can show $m = 3n - 6$ for any plane triangulation.

3 Realizer

In this section we briefly introduce *a realizer*[S90] of a plane triangulation.

Let G be a plane triangulation with three outer vertices r_r, r_b, r_y. We can assume that r_r, r_b, r_y appear on $C_o(G)$ in clockwise order. Those vertices are called *red root, blue root* and *yellow root*, respectively. We denote by V_I the set of inner vertices of G.

A *realizer* R of G is a partition of the inner edges of G into three edge-distinct trees T_r, T_b, T_y satisfying the following conditions (c1) and (c2). See an example in Fig. 3.

(c1) For each $i \in \{r, b, y\}$, T_i is a tree with vertex set $V_I \cup \{r_i\}$.
(c2) For each $i \in \{r, b, y\}$, we regard r_i as the root of T_i, and orient each edge in T_i from a child to its parent. Then at each $v \in V_I$ the edges incident to v appear in clockwise order as follows. See Fig. 2.

(1) exactly one edge in T_r leaving from v.
(2) (zero or more) edges in T_y entering into v.
(3) exactly one edge in T_b leaving from v.
(4) (zero or more) edges in T_r entering into v.
(5) exactly one edge in T_y leaving from v.
(6) (zero or more) edges in T_b entering into v.

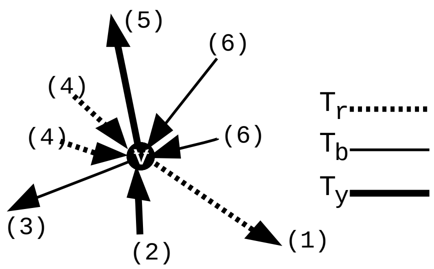

Figure 2: Edges around an inner vertex v.

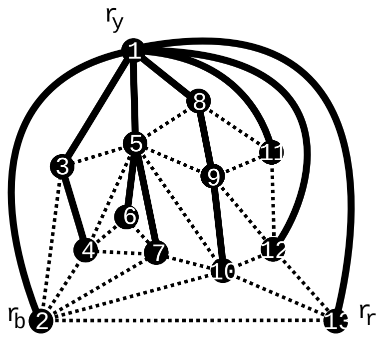

Figure 3: The spanning tree and remaining edges.

Let G be a plane triangulation, and $R = \{T_r, T_b, T_y\}$ be a realizer of G. Again for each $i \in \{r, b, y\}$ we regard r_i as the root of T_i, and orient each edge in T_i from a child to its parent.

Then $T = T_y \cup \{(r_y, r_b), (r_y, r_r)\}$ is a spanning tree of G with root r_y. By preorder traversal of T we assign an integer $i(v)$ for each vertex v. See an example in Fig. 3. Note that $i(r_y) = 1, i(r_b) = 2, i(r_r) = n$ always holds.

We have the following lemma.

Lemma 3.1 *(a) If $e = (u, v)$ is an edge in T_r and orient from u to v, then $i(u) < i(v)$.*
(b) If $e = (u, v)$ is an edge in T_b and orient from u to v, then $i(u) > i(v)$.

Proof. (a) Assume otherwise for the contradiction. Now there is an edge $e = (u, v)$ in T_r and orient from u to v, but $i(u) > i(v)$.

For each $i \in \{r, b, y\}$ let P_i be the path in T_i starting at v and ending at the root r_i. Then by those three paths we partite the plane graph into three regions as follows. Region $\overline{R_r}$: The region inside of $P_y \cup P_b \cup \{(r_b, r_y)\}$. Region $\overline{R_b}$: The region inside of $P_r \cup P_y \cup \{(r_y, r_r)\}$. Region $\overline{R_y}$: The region inside of $P_b \cup P_r \cup \{(r_r, r_b)\}$.

By the condition (c2) of the realizer, vertex u is in $\overline{R_r}$.

By assumption $i(u) > i(v)$ above, the path P in T_y starting at u and ending at the root r_y must contain at least one vertex in $\overline{R_y} \cup \overline{R_b}$. See Fig. 4. Thus P must cross P_b from $\overline{R_r}$ to $\overline{R_y}$.

However, at the crossing point, say vertex y, the condition (c2) of the realizer does not hold. A contradiction.

(b) Similar to (a). Omitted. $\mathcal{Q.E.D.}$

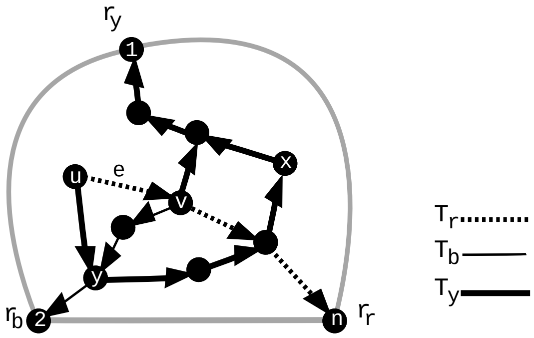

Figure 4: Illustration for Lemma 3.1.

Let $\overline{G_T}$ be the graph derived from G by deleting all edges in the spanning tree T. If $\overline{G_T}$ has an edge (u, v) with $i(u) > i(v)$ then we say v is *a smaller* neighbour of u and u is *a larger* neighbour of v.

We have the following lemma. See Fig. 3.

Lemma 3.2 *(a) Each inner vertex v has at least one smaller neighbour and at least one larger neighbour.*
(b) r_r has at least one smaller neighbour and no larger neighbour.
(c) r_b has no smaller neighbour and at least one larger neighbour.
(b) r_y has neither smaller nor larger neighbour.

Proof. (a)Immediate from Lemma 3.1 and the condition (c2) of the realizer.

Intuitively each inner vertex has one outgoing edge in T_b connecting to one smaller neighbour, and one outgoing edge in T_r connecting to one larger neighbour. See Fig. 3.
(b)(c)(d) Omitted. $\mathcal{Q.E.D.}$

4 Coding

In this section we give our coding scheme for plane triangulations.

Let G be a plane triangulation with a realizer $R = \{T_r, T_b, T_y\}$. Let $T = T_y \cup \{(r_y, r_b), (r_y, r_r)\}$ be a spanning tree of G with root r_y. Assume that by preorder traversal of T each vertex v has an integer label i, as explained in the previous section. Again $\overline{G_T}$ be the graph derived from G by deleting all edges in T. See Fig. 3.

We first encode T into string S_1, then the rest of the graph $\overline{G_T}$ into string S_2. See an example in Fig. 5(a) and (c).

In S_1 each vertex except the root corresponds to a pair of matching parentheses, and if vertex p is the parent of vertex c then the matching parentheses corresponding to p immediately enclose the matching parentheses corresponding to c. See Fig. 5(a).

S_2 consists of $|S_1| - 2$ blocks. See Fig. 5(b). Blocks are hatched alternately to show their boundary. Each block consists of one or more (square) brackets. Each matching brackets corresponds to an edge in $\overline{G_T}$. See Fig. 5(c). Each parenthesis (except for the first and the last one) in S_1 has a corresponding block in S_2. Each open parenthesis "(" in S_1 (except for

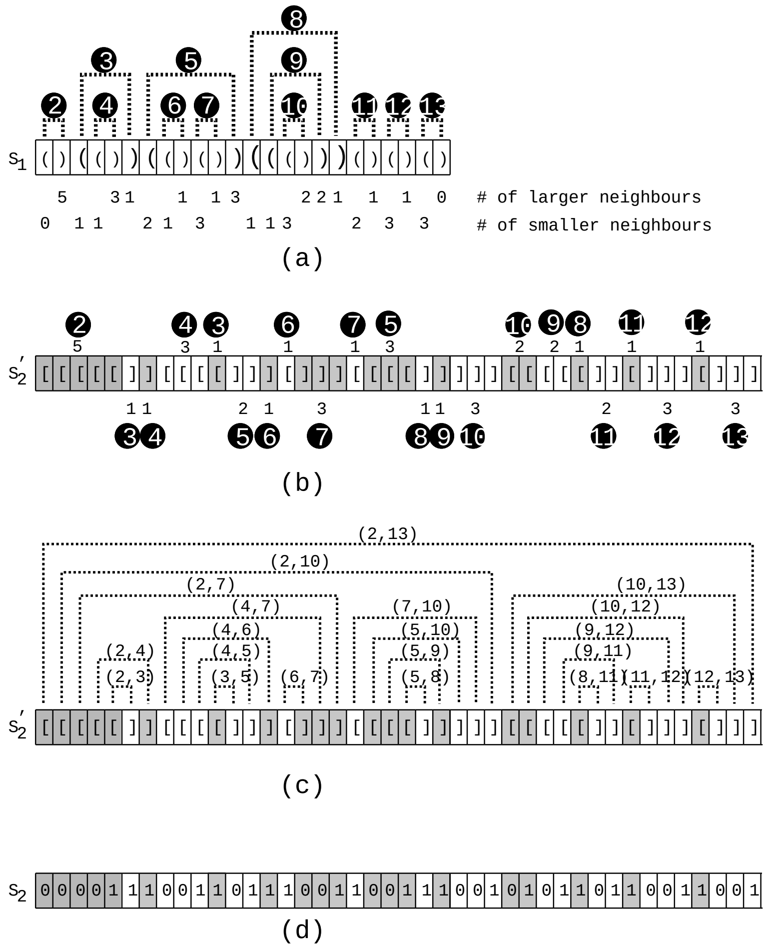

Figure 5: The code.

the first one) has a corresponding block, denoted by $s(v)$, consisting of some "]" 's, and each close parenthesis ")" in S_1 (except for the last one) has a corresponding block, denoted by $l(v)$, consisting of some "[" 's. The length of $s(v)$ is the number of smaller neighbours of v. Thus $s(v)$ consists of $|s(v)|$ of consecutive "]" 's. Similarly, the length of $l(v)$ is the number of larger neighbours of v, and $l(v)$ consists of $|l(v)|$ of consecutive "[" 's.

Since $s(v) \geq 1$ always holds, we can encode the block $s(v)$ as $s(v)-1$ consecutive 0's followed by one 1. See Fig. 5(d). Similarly we encode the block $l(v)$ as $l(v)-1$ consecutive 0's followed by one 1. By the encoding above we can easily recognize the boundary of each block. Note that each block always ends with 1, and 1 is always the end of some block.

Now we explain how to encode given G into S_1 and S_2.

First we encode T as follows. Given a (ordered) trees T we traverse T starting at the root with depth first manner. If we go down an edge then we code it with 1, and if we go up an edge then we code it with 0. Let S_1 be the resulting bit string. The length of S_1 is $2(n-1)$ bits. By regarding the 0 as the open parenthesis "(" and the 1 as the close parenthesis ")", we can regard S_1 as a sequence of balanced parentheses. In S_1 each vertex v except the root r_y correspond to a pair of matching parentheses. Moreover if $i(v)=k$, then v corresponds to the $(k-1)$-th "(" and its matching ")". Note that the root r_y has no corresponding "(".

Next we encode $\overline{G_T}$ as follows. We first copy S_1 above into S_2, and then replace each "(" and ")" by some "]" 's, and "[" 's as follows.

Let $i(v)=k$ and $|s(v)|$ be the number of smaller neighbours of v. If $k \neq 1, 2$ then we replace the $(k-1)$-th "(" by consecutive $|s(v)|-1$ zeros followed by one "1". Similarly, let $|l(v)|$ be the number of larger neighbour of v. If $k \neq 1, n$ then we replace the ")" which matches the $(k-1)$-th "(" by consecutive $|l(v)|-1$ zeros followed by one "1". Note that $s(v) \geq 1$ and $l(v) \geq 1$ always hold for inner vertex v by Lemma 3.2.

The idea is similar to [C01], however by utilizing the claim of Lemma 3.2 we can save two bit for S_2 at each inner vertex.

Now estimate the length of S_1+S_2. We have $|S_1| = 2(n-1)$ and $|S_2| = 2(3n-6-(n-1)) = 4n-10$. Thus $|S_1+S_2| = 2(n-1)+4n-10 = 6n-12 = 2m$.

For example the code in Fig. 5 has lenght $|S_1|+|S_2| = 24+42 = 66$ bits.

We have the following lemma.

Lemma 4.1 *Given a triangulation G we can encode G into S_1+S_2 in $O(n)$ time, where $|S_1+S_2| = 2m$.*

5 Query

In this section we give an efficient algorithm to answer an adjacency and degree queries with a help of an additional string S_A of $o(n)$ bits. We can construct S_A in $O(n)$ time. We also give an algorithm to answer "the clockwise neighbour" query.

We first define several basic operations. Using those basic operations, we can solve each adjacency, degree and clockwise neighbour queries in constant time.

Given a bitstring, *rank(p)*, the rank of the bit at position p is the number of 1's up to and including the position p, and *select(i)* is the position of the i-th 1 in the bitstring.

Given a sequence of balanced parentheses, the following operations are defined. Operation *findclose(p)* computes the position of the close parenthesis that matches the open parenthesis at position p. Operation *findopen(p)* computes the position of the open parenthesis that matches the close parenthesis at position p. Given an open parenthesis at position p, assume q is the position of p's matching close parenthesis, then *enclose(p)* is the position of the open parenthesis which immediately encloses the pair, p and q, of the matching parentheses. Operation *wrapped(p)* computes the number of the positions c_i of open parentheses such that enclose(c_i)=p. Intuitively wrapped(p) is the number of matching parenthesis pairs which are immediately enclosed by the given matching parenthesis pair. The following lemmas are known.

Lemma 5.1 *[MR97, MR01] Given a bitstring of length $2n$, using $o(n)$ auxiliary bits, we can perform the operations rank(p), select(i), in constant time. One can construct the $o(n)$ auxiliary bits in $O(n)$ time.*

Lemma 5.2 *[MR97, MR01] Given a sequence of balanced parentheses of length $2n$, using $o(n)$ auxiliary bits, we can perform the operations findclose(p), findopen(p), enclose(p) in constant time. One can construct the $o(n)$ auxiliary bits in $O(n)$ time.*

Lemma 5.3 *[C01] Given a sequence of balanced parentheses of length $2n$, using $o(n)$ auxiliary bits, we can perform wrapped(p) in constant time. One can construct the $o(n)$ auxiliary bits in $O(n)$ time.*

Then using the basic operations above we can solve an adjacency query in constant time as follows.

Given two integers a and b we are going to decide where G has edge (u, v) such that $i(u) = a$ and $i(v) = b$. We consider the following two cases.

Case 1: $(u, v) \in T$.

For convenience we regard that S_1 is enclosed by a pair of parentheses corresponding to r_y for operation $enclose()$.

Assume that $a < b$. (The other case is similar.) Then $(u, v) \in T$ iff $select(a-1) = enclose(select(b-1))$ in S_1 and we can check this in constant time. Note that since the root r_y has no corresponding "(" thus we need "-1" above. Also note that for operation $select()$ we regard S_1 as a bitstring, and for operation $enclose()$ we regard S_1 as a sequence of balanced parentheses.

Case 2: $(u, v) \in \overline{G_T}$.

Assume that $a < b$. (The other case is similar.) Then $(u, v) \in \overline{G_T}$ iff some "[" in $l(u)$ matches some "]" in $s(v)$. We can check this as follows.

We can recognize the block $l(u)$ in S_2 as follows. First $q = findclose(select(a-1))$ is the position of ")" corresponding to u in S_1. The block corresponding to $l(u)$ starts at position $s_u = select(q-2) + 1$ and ends at $e_u = select(q-1)$ in S_2. Note that S_2 has no block corresponding to $s(2)$, thus we need "-1" above. Similarly we can recognize the block $s(v)$, and assume that the block starts at position s_v and ends at e_v.

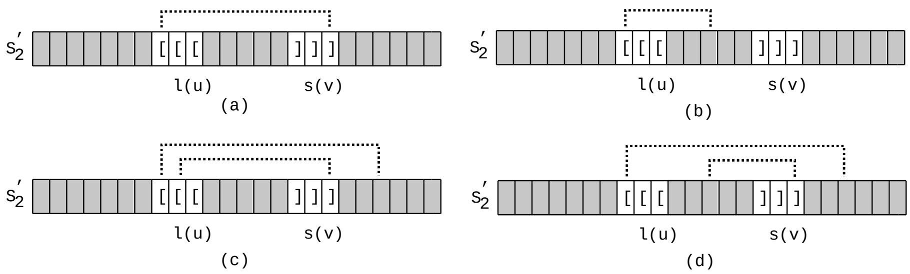

Figure 6: Illustration for the adjacency query.

If $findclose(s_u)$ is located among the block $s(v)$, as shown in Fig. 6(a), then $(u, v) \in \overline{G_T}$. Otherwise, if $findclose(s_u) < s_v$, then $(u, v) \notin \overline{G_T}$. See Fig. 6(b). Otherwise, $findclose(s_u) > e_v$ always holds. If $findopen(e_v)$ is located among the block $l(u)$, then $(u, v) \in \overline{G_T}$. See Fig. 6(c). Otherwise $findopen(e_v) > e_u$ always holds, and $(u, v) \notin \overline{G_T}$. See Fig. 6(d). Thus we can decide whether $(u, v) \in \overline{G_T}$ in constant time.

Also we can solve a degree query in constant time as follows. Given a vertex v we first count the neighbours in T, then the neighbours in $\overline{G_T}$. The sum of them is the degree.

First we count the neighbours in T as follows.

If $i(v) = 1$, then the number n_T of neighbours in T is the number of matching parenthesis pairs which are not enclosed by any matching parenthesis pairs in S_1. For convenience we regard that S_1 is enclosed by a pair of parentheses corresponding to r_y, and compute n_T by so-called "$wrapped(select(0))$". Note that if $i(v) = 1$ then v is the root and has no parent in T.

Otherwise, the number is $1 + wrapped(select(i(v)))$.

Then we count the neighbours in $\overline{G_T}$ as follows. If we can recognize the blocks $s(v)$ and $l(v)$ then the number is $|s(v)| + |l(v)|$.

If $i(v) = 1$ then $|s(v)| + |l(v)| = 0$. If $i(v) = 2$ then $|s(v)| = 0$. If $i(v) = n$ then $|l(v)| = 0$. Otherwise, we can recognize $s(v)$ and $l(v)$ as above, and compute the number in constant time.

Thus we can compute the degree of a given vertex in constant time.

Given two vertex u and its neighbour v with $i(u) = a$ and $i(v) = b$, many plane graph algorithm need to find the neighbour of u succeeding v in clockwise (or counterclockwise) order, since with this query we can (1) trace the boundary of a face, (2) list up the edges around a vertex in clockwise order, and (3) reconstruct G. The neighbour is called *the clockwise neighbour* of u *with respect to* v, and denoted by $cn(u,v)$. We can compute $cn(u,v)$ in constant time, as follows.

Assume that $a > b$. (The other case is similar.) Let $e = (u, cn(u,v))$ be the edge between u and $cn(u,v)$. We have two cases.

Case 1: $(u,v) \in T$.

Then v is the parent of u in T. If $i(u) = n$, then $u = r_r$, $v = r_r$ and $cn(u,v) = r_b$. Otherwise e corresponds to the first "[" in $l(u)$ and its matching "]". With a similar method for adjacent query above we can find the block $l(u)$ and then $cn(u,v)$ in constant time.

Case 2: $(u,v) \in \overline{G_T}$.

In this case e corresponds to some "[" in $l(v)$ and some "] " in $s(u)$.

Assume that e corresponds to the x-th "[" in block $l(v)$ and y-th "] " in $s(u)$. Now we have the following lemma.

Lemma 5.4 *Either* $x = 1$ *or* $x = |l(v)|$ *holds. Either* $y = 1$ *or* $y = |s(u)|$ *holds.*

Proof. Otherwise, $l(v)$ and $s(u)$ has one more matching parenthesis pair "[" and "]", which immediately enclose the matching parenthesis pair corresponding to e. This means G has one more edge between u and v. This contradicts the fact that G has no multi-edge. $\mathcal{Q.E.D.}$

Theorem 5.5 *Given* $S_1 + S_2$*, one can construct an additional string* S_A *of* $o(n)$ *bits in* $O(n)$ *time. Then one can compute adjacency, degree, and clockwise neighbour queries in* $O(1)$ *time, and decode* G *in* $O(n)$ *time.*

References

[CG98] R. C. N. Chuang, A. Garg, X. He, M. Y. Kao and H. I. Lu, *Compact Encodings of Planar Graphs via Canonical Orderings and Multiple Parentheses*, Proc. of ICALP 98, Lecture Notes In Computer Science; Vol. 1443, (1998), pp.118–129.

[C01] Y. T. Chiang, C. C. Lin and H. I. Lu, *Orderly Spanning Trees with Applications to Graph Encoding and Graph Drawing*, Proc. of 12th SODA, (2001), pp.506-515.

[C98] D. R. Clark, *Compact Pat Trees*, PhD thesis, University of Waterloo, (1998).

[H99] X. He, M. Y. Kao and H. I. Lu, *Linear-Time Succinct Encodings of Planar Graphs via Canonical Orderings*, SIAM Journal on Discrete Mathematics, Vol. 12, (1999), pp.317–325.

[H00] X. He, M. Y. Kao and H. I. Lu, *A Fast General Methodology for Information-Theoretically Optimal Encodings of Graphs*, SIAM Journal on Computing, Vol. 30, (2000), pp. 838-846.

[J89] G. Jacobson, *Space-efficient Static Trees and Graphs*, Proc. of 30th FOCS, (1989), pp.549–554.

[KW95] K. Keeler and J. Westbrook, *Short Encodings of Planar Graphs and Maps*, Discrete Applied Mathematics, Vol. 58, (1995), pp.239–252.

[MR97] J. I. Munro and V. Raman, *Succinct Representation of Balanced Parentheses, Static Trees and Planar graphs*, Proc. of 38th FOCS, (1997), pp.118–126.

[MR01] J. I. Munro and V. Raman, *Succinct Representation of Balanced Parentheses and Static Trees*, SIAM J. Comput., Vol. 31, (2001), pp.762–776.

[P03] D. Poulalhon and G. Schaeffer, *Optimal Coding and Sampling of Triangulations*, Proc. of ICALP 03, (2003), pp.1080–1094.

[R00] K. H. Rosen (Eds.), *Handbook of Discrete and Combinatorial Mathematics*, CRC Press, Boca Raton, (2000).

[R99] J. Rossignac, *Edgebreaker: Connectivity compression for triangle meshes*, IEEE Trans. on Visualization and Computer Graphics, Vol. 5, (1999), pp. 47–61.

[S90] W. Schnyder, *Embedding Planar Graphs in the Grid*, Proc. of 1st SODA, (1990), pp.138–147.

[T84] G. Turan, *Succinct Representations of Graphs*, Discrete Applied Mathematics, Vol. 8, (1984), pp.289–294.

[T62] W. Tutte, *A Census of Planar Triangulations*, Canadian Journal of Mathematics, Vol. 14, (1962), pp.21–38.