GENERATING ROOTED TRIANGULATIONS WITHOUT REPETITIONS

*David Avis**

School of Computer Science
McGill University
3480 University
Montréal, Québec, Canada
H3A 2A7

ABSTRACT

We use the reverse search technique to give algorithms for generating all graphs on n points that are two and three connected planar triangulations with r points on the outer face. The triangulations are rooted, which means the outer face has a fixed labelling. The triangulations are produced without duplications in $O(n^2)$ time per triangulation. The algorithms use $O(n)$ space. A program for generating all 3-connected rooted triangulations based on this algorithm is available by ftp.

1. Introduction

Let $G = (V, E)$ be a planar graph with vertex set $V = \{v_1, \ldots, v_n\}$, and let $3 \le r \le n$ be an integer. G is an $r-$*rooted triangulation* if it can be embedded in the plane such that the outer face has labels $\{v_1, \ldots, v_r\}$ in clockwise order, and all interior faces are triangles. A vertex (or edge) on the external face is called *external*, otherwise it is *internal*. All r-rooted triangulations are 2-connected. It is well-known that an r-rooted triangulation is 3-connected if and only if there does not exist an edge between two non-consecutive vertices on the outer face. It follows that all 3-rooted triangulations are 3-connected. Let m be the number of edges in G. If follows from Euler's formula that $m = 3n - 3 - r$. We denote the *degree* of a vertex v by $d(v)$.

Two r-rooted triangulations are isomorphic if there is an edge preserving isomorphism between the vertex sets of the two triangulations that preserves the labelling of the outer face. For given n, r, let $f(n, r)$ be the number of non-isomorphic 2-connected triangulations, and let $g(n, r)$ be the number of non-isomorphic 3-connected triangulations. Observe that $f(n, 3) = g(n, 3)$, $n \ge 3$. W. Tutte[11] found a closed formula for $g(n, r)$. For $n \ge 5$,

$$g(n, 3) = \frac{2}{(n-2)!}(3n-6)(3n-5)\ldots(4n-11).$$

For $r \ge 4$, $n \ge r + 2$,

$$g(n, r) = \frac{3(r-1)!(r-4)!}{(3n-6)!} \sum_{j=0}^{min(n-r-1,\, r-3)} \frac{(4n-r-8-j)!(r-1+j)(r-3-3j)}{j!(j+1)!(r-3-j)!(r-1-j)!(n-r-j-1)!}.$$

* Research supported by N.S.E.R.C. grant number A3013, F.C.A.R. grant number EQ1678, and a bilateral exchange from J.S.P.S./N.S.E.R.C.

W. Brown[5] found a closed formula for $f(n, r)$. For $n \geq r \geq 3$

$$f(n, r) = \frac{2(2r-3)!(4n-2r-5)!}{(r-1)!(r-3)!(n-r)!(3n-r-3)!} .$$

The efficient generation of unrooted triangulations has received some attention in the literature. This appears to be harder than generating all rooted triangulations, and isomorphism testing is required by current algorithms. This means that all non-isomorphic triangulations generated must be stored. R. Bowen and S. Fisk[4] describe a method of generating all triangulations of the sphere. M. B. Dillencourt[6] gives an algorithm for the equivalent dual problem of generating all simplicial polyhedra in 3 dimensions. It is an open problem whether reverse search can be used to generate unrooted triangulations without isomorphism testing.

In this paper we show how to generate all 3-connected r-rooted triangulations in $O(n^2 g(n, r))$ time and all 2-connected r-rooted triangulations in $O(n^2 f(n, r))$ time. We will use the reverse search method developed by K. Fukuda and the author[1], [2]. A feature of the reverse search method is that no isomorphism test is required, and the triangulations need not be stored. The algorithm uses only $O(n)$ space. We begin by reviewing informally the reverse search method in the context of our application. It will be assumed for the rest of this section that we wish to generate all 3-connected r-rooted triangulations on n points, for given fixed n and r. In the next section we give the details of the required procedures for the 3-connected case. Section 3 contains a description of the reverse search procedure, the data structure required and an analysis of time and space complexity. Section 4 describes the modifications necessary to generate 2-connected triangulations. Finally in Section 5 we discuss some computational experience, open problems, possible improvements and explain how a copy of the program may be obtained.

The reverse search method is a technique for generating all vertices of a graph whose edges are given implicitly by an oracle. Let $H = (T, U)$ be such a graph. In our application, each vertex in T corresponds to a r-rooted triangulation with given fixed n and r. U is the edge set of adjacent vertices in T. In our application, two r-rooted triangulations are adjacent if they differ by exactly one edge. The reverse search method works by finding a spanning tree in the graph H. To do this we first fix some given triangulation with the required parameters n and r. Call the vertex in H corresponding to this triangulation the *target* t^*. We now define a *local search* procedure, L, that given any vertex t of H defines a unique adjacent vertex in H, with the property that repeated application of L defines a path in H to the target. In other words, $(t, L(t))$ is an edge in U and $L^k(t) = t^*$ for some finite integer k. The path generated consists of a sequence of r-rooted triangulations that differ in exactly one edge and ends with the target triangulation. The set of all such paths clearly forms a spanning tree in H.

The reverse search procedure is initiated at the target t^* and constructs a spanning tree of H by *reversing* the local search procedure. To do this we first generate all neighbours in H of any given vertex $t \in T$, in some given order. This is done by an *adjacency oracle*. Using the adjacency oracle at t^* we consider neighbours of t^* until we find a neighbour t such that $L(t) = t^*$. We now replace t^* by t and use the adjacency oracle to find (if possible) a neighbour s of t such that $L(s) = t$. If such a vertex s exists we move to s and continue. If we reach a node t for which no such neighbour s exists, we backtrack by computing the parent $u = L(t)$ of t. We now continue from u using the adjacency oracle to give the next neighbour of u in order after t.

In order to give a formal description of a reverse search procedure it is necessary to specify:

(a) the target triangulation;

(b) the adjacency oracle;

(c) the local search procedure.

These are specified for the 3 and 2 connected cases in Sections 2 and 4, respectively.

2. 3-Connected Triangulations

In this section we describe formally how to generate all 3-connected r-rooted triangulations with n points. To avoid trivial cases we assume $n > r \geq 3$.

2.1. Target Triangulations

For each $n > r \geq 3$ we define a target triangulation $E^*_{n,r}$. Figure 2.1(a) shows the target triangulation $E^*_{5,3}$ for $n = 5$ and $r = 3$. This graph is an example of a *prism*. The target triangulation $E^*_{7,4}$ for $n = 7$ and $r = 4$ is shown in Figure 2.1(b) In general, we first build a *wheel* on the external face $v_1, \cdots, v_r$ with centre v_{r+1}. Then we build a *prism* in the triangle $v_{r+1} v_r v_{r-1}$. The edge list of $E^*_{n,r}$ for $n > r \geq 3$ is

$$v_1 v_2,\ v_2 v_3, \ldots,\ v_{r-1} v_r,\ v_1 v_r,$$

$$v_1 v_{r+1},\ v_2 v_{r+1}, \ldots,\ v_r v_{r+1},$$

for the wheel, and

$$v_{r-1} v_{r+2},\ v_{r-1} v_{r+3}, \ldots,\ v_{r-1} v_n,$$

$$v_r v_{r+2},\ v_r v_{r+3}, \ldots,\ v_r v_n,$$

$$v_{r+1} v_{r+2},\ v_{r+2} v_{r+3}, \ldots,\ v_{n-1} v_n.$$

for the prism.

2.2. Adjacency Oracle

Let $G = (V, E)$ be an r-rooted 3-connected triangulation and let $e = v_a v_b \in E$ be an internal edge, that is, an edge that is not on the external face. Since e is internal it bounds two triangles, say $v_a\ v_b\ v_c$ and $v_a\ v_b\ v_d$. If $e' = v_c v_d$ is not an edge in E we say that e is *transformable* and define the notation

$$E \,\Delta\, e = E - e + e'.$$

Note that once a transformable edge e is chosen, the edge e' is well defined, and need not be specified explicitly. It is easy to see that $G' = (V, E \,\Delta\, e)$ is also an r-rooted triangulation. We say that G and G' are adjacent triangulations. The transformation of E to $E \,\Delta\, e$ is well-known, and is called a *diagonal transformation*. When edge e is *transformable*, we say that we can *flip* edge e. G' is 3-connected if the edge e' does not join two vertices of the outer face. An edge transformation of edge $v_1 v_5$ of $E^*_{7,4}$ (Figure 2.1(b)) produces a 2-connected triangulation, for example.

We now define an adjacency oracle that gives all adjacent triangulations to a given triangulation G. We assume that the edge list of G is stored in some given order and let

$E = (e_1, e_2, \ldots, e_m)$ be the ordered edge list. For $j = 1, \ldots, m$ we define the adjacency oracle $Adj(E, j)$ by

$$Adj(E, j) = \begin{cases} E \, \Delta \, e_j & \text{if } e_j \text{ is transformable} \\ \varnothing & \text{otherwise.} \end{cases}$$

In the ordered edge list $E \, \Delta \, e_j$, the edge replacing e_j is stored in the j-th position of the edge list, the other edges remain in their original order. The adjacency oracle *Adj* defines a graph $H = (T, U)$ on the set of 3-connected r-rooted triangulations for given n. The vertices T of H correspond to the edge sets of the triangulations. There is an edge in U between the edge sets E and E' if and only if

$$E' = Adj(E, j) \quad \text{for some } j,\ 1 \le j \le m.$$

In the next section we show that H is connected and give a local search procedure for H.

2.3. Local Search Procedure

For $r = 3$, the target triangulation in Section 2.1 and an implicit description of the configuration graph H of Section 2.2 was given by K. Wagner[12] and also appears in the book of O. Ore[8]. Wagner proved that H is connected when $r = 3$. A somewhat simpler proof is contained in Ore and this proof is suitable for defining an efficient local search procedure. We give a proof based on a similar idea for the general case $r \ge 3$. We begin with a basic lemma which is illustrated in Figures 2.2(a) and (b). We say that a vertex v has *consecutive* neighbours $t_1, t_2, \ldots, t_k$ if these neighbours occur in consecutive counter-clockwise order in the unique planar embedding of the r-rooted triangulation.

Lemma 2.1

Let s be a vertex in an r-rooted triangulation G with four consecutive neighbours t_1, t_2, t_3, t_4, where t_1 is on the external face and t_2 is not. Then either st_3 is transformable or t_2t_4 is an edge of G which is transformable. The edge transformation preserves 3-connectivity.

Proof: Since t_1, t_2, t_3, t_4 are consecutive neighbours of s, the edge st_3 bounds the triangles $s\ t_2\ t_3$ and $s\ t_3\ t_4$. If t_2t_4 is not an edge then st_3 is transformable. Otherwise since t_1 is external, t_3 is inside the triangle $s\ t_2\ t_4$ and adjacent to these three vertices. Since t_2 is internal, the edge t_2t_4 is internal and bounds this triangle and some other triangle $x\ t_2\ t_4$. Now x is not adjacent to t_3 so t_2t_4 is transformable.

Suppose G is 3-connected. It suffices to show the new edge contains an internal vertex. The edge transformation produces either the new edge t_2t_4 or the new edge t_3x. In the first case t_2 is internal by hypothesis and in the second case t_3 is internal since it lies inside triangle st_2t_4. In both cases 3-connectivity is preserved. □

Before formally defining the local search algorithm, we will illustrate it on the example G_1 in Figure 2.3. The idea is to compare vertices $v_1, v_2, v_3, \ldots$ in G_1 with the corresponding vertices in the target, $E^*_{7,4}$, shown in Figure 2.1(b). When the first vertex is found that differs from the target, the lemma is applied. This is repeated until the target is reached. Comparing the two triangulations G_1 and $E^*_{7,4}$ we see that $d(v_1) = 3$ in G_1, matching its counterpart in the target, but

$d(v_2) = 4$ does not. We apply the lemma with $s = v_2$, $t_1 = v_1$, $t_2 = a$, $t_3 = b$ and $t_4 = v_3$. Since av_3 is an edge, $v_2 b$ is not transformable so we flip av_3 getting G_2. In G_2, v_1 again has the correct degree and v_2 has degree 4. Applying the lemma again it is now possible to flip $v_2 b$ reducing the degree of v_2 to three and obtaining G_3. In G_3 the degrees of both v_1 and v_2 are both three so we have a wheel with centre a, external face $v_1, \ldots, v_4$ and all remaining vertices in the triangle $a\ v_3\ v_4$. We now transform this triangle into the target prism shown in Figure 2.1(a). For this purpose imagine vertices v_1 and v_2 along with incident edges have been deleted.

Inside triangle $a\ v_3\ v_4$ vertex a has degree 4 so we apply the lemma with $t_1 = v_4$, $t_2 = c$, $t_3 = b$ and $t_4 = v_3$, getting the transformable edge cv_3. The resulting triangulation G_4 has similar structure and the lemma is applied again to the same vertex set giving the transformable edge ab. This diagonal transformation leads to the target $E^*_{7,4}$, with $v_5 = a$, $v_6 = c$ and $v_7 = b$.

The above method is stated formally as procedure *LocalSearch* in Figure 2.4. Let E be the edge list of a 3-connected r-rooted triangulation on n points that is not the target triangulation. *LocalSearch*(E, n, r) returns a transformable edge $e \in E$. The next theorem shows that repeated application of this procedure leads to the target triangulation.

Theorem 2.1

(a) *If E is a 3-connected triangulation, LocalSearch(E, n, r) returns a transformable edge $e \in E$ which preserves 3-connectivity.*

(b) *By repeated application of LocalSearch to the transformed edge set $E \,\Delta\, e$ the target triangulation $E^*_{n,\,r}$ is reached.*

Proof: For part (a) of the theorem, we apply Lemma 2.1. If e is chosen in the first part of the procedure, s and t_1 are on the external face and $d(s) = 4$. By 3-connectivity, t_2 is an internal vertex. If e is chosen in the second part of the procedure, t_2 is inside the triangle $sv_{r-1}v_r$ and so it is internal. In both cases the lemma applies, the edge chosen is transformable and 3-connectivity is preserved.

For part (b), we first show that E is transformed to a triangulation with $d(v_i) = 3$ for $i = 1, \ldots, r-2$. Indeed, suppose i is the smallest index in this range such that $d(v_i) > 3$. Vertex v_i is selected in the part of the procedure. Since v_i is on the external face and the triangulation is 3-connected, the second consecutive neighbour of v_i must be internal and so Lemma 2.1 can be applied. Suppose e is the transformable edge provided by the lemma and consider the triangulation $E' = E \,\Delta\, e$. Either $d(v_i)$ has been decreased by one, or *LocalSearch*(E', n, r) returns an edge e' such that the triangulation $E' \Delta e'$ has this property.

We must show that the diagonal transformation does not increase the degrees of $v_1, \ldots, v_{i-1}$. There are two cases depending on the choice of edge in the lemma. First, if st_3 is transformable, the new edge is $t_2 t_4$. Since $s = v_i$ and $t_1 = v_{i-1}$ are external, 3-connectivity implies that t_4 is either internal or $t_4 = v_{i+1}$. In either case the transformation does not increase the degrees of $v_1, \ldots, v_{i-1}$. In the second case, $t_2 t_4$ is an edge, and the new edge is xt_3. Suppose $x = v_k$ for some $1 \le k \le i-1$. Now t_4 must also be external, for otherwise v_k is adjacent to v_{k-1}, v_{k+1}, t_2 and t_4 before the transformation, violating the condition that $d(v_k) = 3$. But if t_4 is external, we have $t_4 = v_{i+1}$ by 3-connectivity. Since t_4 is also adjacent to $x = v_k$, we must have $t_4 = v_r$ so $i = r-1$ contradicting the choice of i by *LocalSearch*. Therefore x is either internal or external with index greater than i and the degrees of $v_1, \ldots, v_{i-1}$ are preserved. It follows that repeated application of

LocalSearch produces a triangulation with $d(v_i) = 3$ for $i = 1, \ldots, r-2$.

We now have a triangulation that is a wheel with external face $v_1, \ldots, v_r$, with centre, say, s and with the remaining vertices in the triangle $s\ v_{r-1}\ v_r$. Let us label s as v_{r+1}. The triangulation inside this triangle is to be transformed into a prism. Firstly, if s is adjacent to more than one internal vertex, we apply Lemma 2.1 to the 4 consecutive neighbours of s in counter-clockwise order starting from v_r. Note that since $v_{r-1}v_r$ is an edge of the external face, the second and third consecutive neighbours of s are internal. If we do a diagonal transformation on the edge returned, either the degree of s is reduced by one, or this happens on the following application of *LocalSearch* and subsequent diagonal transformation. We continue in this way, until s is adjacent to one internal vertex, say, a. Let us label a as v_{r+2} It follows that a is adjacent to both v_{r-1} and v_r. Setting *lasts* $= s$ and $s = a$ we now repeat essentially the same steps until the new vertex s has exactly one internal vertex (say $a = v_{r+3}$) that is different from *lasts*. We move down the chain $v_{r+1}, v_{r+2}, \cdots$ until a vertex is found that differs from the corresponding vertex in the target, and apply the above procedure.

It can be verified that once the vertices $v_{r+1}, v_{r+2}, \ldots$ have degrees corresponding to the target they are never involved in further edge transformations. Therefore the procedure is finite and the theorem is proved. □

3. Implementation of Reverse Search Procedure

In this section we give the details on how the reverse search technique is used to generate triangulations. We begin by giving the procedure *ReverseSearch* which is the standard reverse search procedure, essentially as given in [2]. The reader unfamiliar with reverse search is referred to this paper for further description, formal proofs of correctness and complexity analysis. However, reference to the informal description in Section 1 and the code *ReverseSearch* shown in Figure 3.1 should give the essential idea.

We now give a simple data structure that allows *LocalSearch* and *Adj* to be implemented in $O(n)$ time. After a diagonal transformation, the data structure can be updated in $O(n)$ time. This leads to an implementation of *ReverseSearch* in $O(n^2 g(n, r))$ time and $O(n)$ space. The data structure used is the $doubly-connected-edge-list$, or DCEL, of D.E. Muller and F. P. Preparata[9], a description of which can be found in[10]. Each edge ab of the triangulation is represented by a corresponding *edge node* in the DCEL. The edge node for ab contains the vertex labels a and b and two pointers:

- a pointer to the edge node corresponding to the edge ac which is next in counter-clockwise order about a after ab, and

- a pointer to the edge node corresponding to the edge bd which is next in counter-clockwise order about b after ab.

The edge nodes are stored in an array. Each entry of the array contains two data fields for the vertex labels and two pointer fields containing the array indices for the two edge nodes described above. The data structure for $E^*_{5,3}$ is shown in Figure 3.2. This data structure has the following properties:

```
procedure LocalSearch(E, n, r);
 /* return a transformable edge or null if E = E*_{n,r} */

 /* create a wheel with centre v_{r+1} */

i := 1;
while  d(v_i) = 3 and i ≤ r − 2
        i := i + 1;
endwhile
if i ≤ r − 2 then                /*  d(v_i) > 3/*
        let t_1 := v_{i−1}, t_2, t_3, t_4 be consecutive neighbours of v_i in counter-clockwise order;
        if t_2t_4 is not an edge then return(st_3)
           else return (t_2t_4);
endif

 /* Transform the triangle v_{r+1} v_{r−1} v_r to a prism */

let s := the vertex adjacent to v_1, . . . , v_r;
lasts := v_1;
while s is adjacent to exactly one internal vertex a that is not lasts
                lasts := s;    s := a;
endwhile

if  d(s) = 3 return (∅);    /* E = E*_{n,r} */

 /* We have found the first vertex where E is different from the target*/

let t_1 := v_r, t_2, t_3, t_4 be consecutive neighbours of s in counter-clockwise order;
if t_2t_4 is not an edge then return(st_3)
 else return (t_2t_4);
```

Figure 2.4 Local Search Procedure for 3-connected Triangulations

(P1) From an edge node *ab*, the edge nodes of all of the edges of the triangles *abc* and *abd* can be obtained in constant time.

(P2) From an edge node *ab*, the edges adjacent to *a* (or *b*) may be obtained in counterclockwise order, starting from *ab*, at a cost of constant time per edge.

Consider first the implementation of the adjacency oracle $Adj(E, j)$. The edge counter j is used as an index into the array storing the edge nodes. For given j, an edge *ab* is located in constant time. By P1, the two triangles *abc* and *abd* bounded by *ab* can also be determined in

procedure *ReverseSearch*(n, r);

$E := E^{*}_{n,r}$;   /* start at target triangulation */
$j := 0$;   /* adjacency counter */
repeat
    while $j < 3n - 3 - r$ /* maximum number of edges */
        $j := j + 1$;
        $E' := Adj(E, j)$;
        if $E \neq \emptyset$ **then**
            $e_k := LocalSearch(E', n, r)$;
                if $e_k = e_j$ **then**
                    /*go down reverse search tree*/
                    $E := E \,\Delta\, e_k$; $j := 0$;
                endif
        endif
    endwhile
    if $E \neq E^{*}_{n,r}$ **then**   /*backtrack*/
        $e_k := LocalSearch(E, n, r)$;
        $j := k$;   /* restore adjacency counter */
        $E := E \,\Delta\, e_k$;
    endif
until $E = E^{*}_{n,r}$ **and** $j = 3n - 3 - r$ /* all edges explored from target */

Figure 3.1   Pseudo-code for ***ReverseSearch***

constant time. A sequential search of the edge nodes is used to see if *cd* is an edge. If so, the empty set is returned. If not, a diagonal transformation is performed on edge *ab*. This can be done in constant time. The updates to DCEL are shown in Figure 3.3(a) and (b). It is assumed that *d* is the next neighbour of *a* in counter-clockwise order after *b*, and fields not shown are unchanged.

index	**vertex 1**	**vertex 2**	**next vertex 1**	**next vertex 2**
r1	*a*	*b*	r3	r4
r2	*a*	*c*	r1	
r3	*a*	*d*		r5
r4	*b*	*c*		r2
r5	*b*	*d*	r1	

Figure 3.3(a) DCEL before diagonal transformation on ***ab***

index	vertex	vertex	next	next
	1	2	vertex 1	vertex 2
r1	*c*	*d*	r2	r5
r2	*a*	*c*	r3	
r3	*a*	*d*		r1
r4	*b*	*c*		r1
r5	*b*	*d*	r4	

Figure 3.3(b) DCEL after diagonal transformation on *ab*

Observe that if edge *ab* is transformable and its location in DCEL is known, then the transformation takes $O(1)$ time. Testing transformability takes $O(n)$ time since the edges are not stored in order in DCEL.

Next consider the implementation of *LocalSearch*. This procedure finds the first vertex with degree greater than that of the target, where the vertices are checked in order around the outer face and then from the centre of the prism to the external face. By property P2, this can be done in time proportional to the number of edges, and the traversal of the vertices in the given order can easily be achieved with the DCEL structure. Once the first vertex of the triangulation differing from the target is found, this becomes vertex s of Lemma 2.1. A scan around s produces the vertices t_1, t_2, t_3, t_4. A linear scan of the edge nodes determines which of the cases of the lemma apply. The total time required by *LocalSearch* is therefore $O(n)$.

We can now analyze *ReverseSearch*. The while loop of the repeat block dominates the computation. For each triangulation and each edge in the triangulation it is necessary to call *Adj* and possibly *LocalSearch*. The total time for this step is therefore $O(n^2 g(n, r))$.

4. 2-Connected Triangulations

In this section we describe the modifications needed to produce all 2-connected triangulations. To avoid cases covered already we may assume $n \geq r \geq 4$. The modifications required are new target triangulations and a slightly different local search procedure. The adjacency oracle described in Section 2.2 is valid without change.

Figure 4.1 shows the target triangulation $F^*_{6,5}$. In general the target $F^*_{n,r}$ is specified by

(a) creating a star-shaped triangulation of the outer face from vertex v_r, and

(b) creating a prism in the triangle $v_{r-2}\ v_{r-1} v_r$ with the remaining $n - r$ points, if any.

The edge list of $F^*_{n,r}$ for $n \geq r \geq 4$ is

$$v_1 v_2,\ v_2 v_3, \ldots,\ v_{r-2} v_{r-1},$$

$$v_1 v_r,\ v_2 v_r, \ldots,\ v_{r-1} v_r,$$

for the outer face, and, when $n > r$,

$$v_{r-2} v_{r+1},\ v_{r+1} v_{r+2},\ v_{r+2} v_{r+3}, \ldots,\ v_{n-1} v_n$$

$$v_{r-1} v_{r+1},\ v_{r-1} v_{r+2}, \ldots,\ v_{r-1} v_n,$$

index	vertex	vertex	next	next
	1	2	vertex 1	vertex 2
1	v_1	v_2	2	5
2	v_1	v_3	3	4
3	v_1	v_4	1	7
4	v_2	v_3	1	8
5	v_2	v_4	6	3
6	v_2	v_5	4	9
7	v_3	v_4	2	9
8	v_3	v_5	7	6
9	v_4	v_5	5	8

Figure 3.2 DCEL data structure for $E^*_{5,3}$

$$v_r v_{r+1},\ v_r v_{r+2}, \ldots,\ v_r v_n,$$

for the prism.

The local search procedure for generating 2-connected triangulations is based on the same idea as described in Section 2.3 for 3-connected triangulations. Let E be the edge list of a triangulation that is not the target $F^*_{n,r}$. We check the vertices sequentially about the external face $v_1, v_2, \ldots, v_{r-3}$ reducing the degree of vertex v_1 to 2, and the other degrees to 3. When this has been accomplished, the outer face has been triangulated as a star from vertex v_r and all internal vertices lie inside the triangle $v_{r-2}\ v_{r-1} v_r$. We now transform this triangle into a prism as described in Section 2.3. The transformation of the outer face to the target is based on the following lemma.

Lemma 4.1

For $i = 1, 2, \ldots, r-3$, if v_i is adjacent to consecutive vertices v_r, t_1, t_2, then either $v_i t_1$, $v_i t_2$ or $v_r t_2$ is a transformable edge.

Proof: If $v_i t_1$ is not transformable, $v_r t_2$ must be an edge, and t_1 lies inside the triangle $v_i v_r t_2$. First suppose $t_2 \neq v_{i+1}$ (Figure 4.2(a)). Then v_i has at least one more consecutive neighbour after t_2. Therefore $v_i t_2$ is an internal edge bounding triangles $v_i t_1 t_2$ and $v_i t_2 a$ for some vertex a. Since a cannot be adjacent to t_1, edge $v_i t_2$ is transformable. Otherwise $t_2 = v_{i+1}$ and since $i \leq r-3$, $v_{i+1} v_r$ is an internal edge bounding triangles $t_1 v_{i+1} v_r$ and $b v_{i+1} v_r$, for some vertex b (Figure 4.2(b)). Vertices b and t_1 cannot be adjacent, so $v_r t_2$ is transformable. □

Based on this Lemmas 2.1 and 4.1 we present in Figure 4.3 the procedure *LocalSearch*2 which transforms and 2-connected rooted triangulation to the target. An application of the procedure is shown in Figure 4.1. Analagous to Theorem 2.1 we have the following result.

Theorem 4.1

(a) *If E is a 2-connected triangulation, LocalSearch*$2(E, n, r)$ *returns a transformable edge e.*

(b) *By repeated application of LocalSearch*2 *to the transformed edge set* $E \Delta e$ *the target triangulation* $F^*_{n,r}$ *is reached.* □

To prove the theorem, Lemma 4.1 is applied when the edge is selected in the first step of *LocalSearch*2. When the edge is selected in the second part, Lemma 2.1 is applied. The proof is similar to Theorem 2.1 and the details are omitted.

---

procedure *LocalSearch*$2(E, n, r)$;

/* Assume $n \geq r \geq 4$ */
/* Create a star triangulation from v_r */

if the degree of $v_1 > 2$ or the degree of $v_i > 4$ for some $i = 2, 3, \ldots, r-3$ **then**
choose the smallest such i and suppose v_i is adjacent to consecutive vertices v_r, t_1, t_2;
if $v_r t_2$ is not an edge **then return**$(v_i t_1)$;
if $t_2 \neq v_{i+1}$ **then return**$(v_i t_2)$;
else return$(v_r t_2)$;
endif

/* Transform the triangle $v_{r-2}\ v_{r-1}\ v_r$ to a prism */

$s = v_{r-2}$;
if s is adjacent to exactly one internal vertex a **then**
lasts $= s$; $s = a$;
while s is adjacent to exactly 2 internal vertices *lasts* and a
lasts $= s$; $s = a$;
endwhile
endif

/* We have found the first vertex where E is different from the target*/

Suppose $t_1 = v_r, t_2, t_3, t_4$ are consecutive vertices adjacent to s in counter-clockwise order;
if $t_2 t_4$ is not an edge **then return**$(s t_3)$
else return $(t_2 t_4)$;

Figure 4.3 Local Search for 2-connected Triangulations

Using *LocalSearch*2 and $F^*_{n,r}$ in place of *LocalSearch* and $E^*_{n,r}$, respectively, in *ReverseSearch* we obtain an algorithm for generating all 2-connected triangulations. This algorithm can be implemented using the DCEL data structure described in Section 3. Using an analysis similar to that given in Section 3, we can show that *LocalSearch*2 also can be implemented in $O(n)$ time. We conclude that the modified *ReverseSearch* procedure produces all 2-connected triangulations in $O(n^2 f(n,r))$ time and $O(n)$ space.

5. Concluding Remarks

We have presented efficient algorithms to generate all 2 and 3-connected rooted triangulations. The algorithm for generating 3-connected triangulations was programmed using essentially the method described here with a slightly different data structure. The program is available by FTP to mutt.cs.mcgill.ca. Please use the login name ftp, give your full email address as password and change to directory *pub/tri*. The program is written in C and called $3tri.c$. It was tested for all n and r in the range $3 \le r < n \le 12$, and the number of triangulations matched the Tutte formula for $g(n,r)$. The largest problem solved, with $n = 13$, $r = 3$ and $g(13,3) = 6,369,883$, took 270 minutes on a Silicon Graphics workstation. The program has been used to settle two questions involving triangulations.

Using a list of 3-rooted triangulations on 9 points, Binhai Zhu verified that each triangulation has a set of two dominating edges. This means that each triangle of the triangulation contains at least one vertex which is an endpoint of one of the edges. This results in an improvement of the lower bound on the number of edge guards required to guard a polyhedral terrain given in [3].

Ferran Huratado[7] asked the author if there exist eulerian 3-rooted triangulations for $n \ge 11$, for n odd and n not divisible by 3. It was known that none exist for n =4,5 and 7[7]. These results were verified, and it was found that for n =6,8,9,10,11,12, and 13 there are respectively 1,3,7,15,63,168, and 561 Eulerian 3-rooted triangulations.

The reverse search method used to generate all r-rooted triangulations is the so-called naive method. The dominant step in the algorithm is going down the reverse search tree, which is essentially testing whether

$$e_j = LocalSearch(E \Delta e_j, n, r) \tag{5.1}$$

for each edge e_j in triangulation E. In the naive method, if e_j is transformable, the transformation is made and then *LocalSearch* is applied, in time $O(n)$, to determine if e_j is the chosen edge. In many applications of reverse search, this test can be done more efficiently [2]. In our case, many candidate edges e_j can be rejected without computing $E \Delta e_j$ and calling *LocalSearch*. It is an open problem whether the test (5.1) can be done in $O(1)$ time. The constants implied by the big-O notation are important, since this problem is feasible only for small values of n.

As mentioned in the introduction, another open problem is to apply reverse search to compute all non-isomorphic unrooted triangulations. More generally it would be of interest to generate all 3-connected planar graphs, or dually, all distinct combinatorial types of 3 dimensional polyhedra without repetitions.

6. Acknowledgements

The author is grateful to Antoine Deza, Komei Fukuda, Bill Lenhart, S. Negami, G. Toussaint and the students of his graduate class in Combinatorial Algorithms at McGill for discussions on this topic. The author thanks Masakazu Kojima for providing facilities in his laboratory at

Tokyo Institute of Technology, where the programming was done.

References

1. D. Avis and K. Fukuda, "A Pivoting Algorithm for Convex Hulls and Vertex Enumeration of Arrangements and Polyhedra," *Discrete and Computational Geometry*, vol. 8, pp. 295-313, 1992.
2. D. Avis and K. Fukuda, "Reverse Search for Enumeration," *Research Report No. 92-5*, GSSM, University of Tsukuba, April 1992. Discrete Applied Math (to appear)
3. P. Bose, T. Shermer, G. T. Toussaint, and B. Zhu, "Guarding Polyhedral Terrains," *Technical Report SOCS 92.20*, McGill University, 1992.
4. R. Bowen and S. Fisk, "Generation of Triangulations of the Sphere," *Math. Comp.*, vol. 21, pp. 250-252, 1967.
5. W. G. Brown, "Enumeration of Triangulations of the Disk," *Proc. London Mathematical Soc.*, vol. 14, pp. 746-768, 1964.
6. M. B. Dillencourt, *Polyhedra of Small Order and Their Hamiltonian Properties,* Technical Report 92-91, University of California, Irvine, Information and Computer Science, September, 1992.
7. F. Hurtado. Private Communication
8. O. Ore, in *The four-color problem*, Academic Press, 1967.
9. F. P. Preparata and D. E. Muller, "Finding the Intersection of Two Convex Polyhedra," *Theoretical Computer Science*, vol. 7, pp. 217-236, 1978.
10. F. P. Preparata and M. I. Shamos, *Computational Geometry,* Springer-Verlag, 1985.
11. W.T. Tutte, "A Census of Planar Triangulations," *Canadian J. Math.*, vol. 14, pp. 21-38.
12. K. Wagner, "Bemerkungen zum Vierfarbenproblem," *J. ber. Deut. Math. Ver.*, vol. 46, pp. 26-32, 1936. Abt. 1