Generating Rooted Triangulations with Minimum Degree Four

(An extended abstract.)

David Avis and Chiu Ming Kong

School of Computer Science
McGill University, Montreal
{avis,min}@cs.mcgill.ca

1 Introduction

A graph is a triangulation if it is planar and every face is a triangle. A triangulation is rooted if the external triangular face is labelled. Two rooted triangulations with the same external face labels are isomorphic if their internal vertices can be labelled so that both triangulations have identical edge lists.

In this article, we show that in the set of rooted triangulations on n points with minimum degree four, there exists a target triangulation E_n^* such that any other triangulation $E_n \neq E_n^*$ in the set can be transformed to E_n^* via a finite sequence of single and double diagonal transformations. Using this result with the reverse search technique, we present an algorithm for generating all non-isomorphic rooted triangulations on n points with minimum degree four. The triangulations are produced without repetitions in $O(n^2)$ time per triangulation. The algorithm uses $O(n)$ space.

2 Background

2.1 Maximal Planar Graphs and Diagonal Transformation

A simple planar graph is **maximal planar** if and only if it is triangulated. Hence a **triangulation** of a set of points in the plane is a maximal planar graph. In a **maximal planar graph**(**MPG**) $G = (V, E)$, $|E| = 3|V| - 6$.

An edge in a triangulation is **external** if it is contained in exactly one triangle (not the external face) of the triangulation. The vertices of the external edges are **external vertices**. Vertices that are not external are **internal vertices**. An edge in a triangulation is **internal** if it is contained in exactly two triangles (neither is the external face) of the triangulation. Such an edge **bounds** the two triangles.

The objects we want to generate are **rooted triangulations with minimum degree four** on n points (given n), which, hereafter, will be referred to as **rooted maximal planar graphs with minimum degree four** (**rooted MPG4**).

In a maximal planar graph G, let $\triangle abc$ and $\triangle abd$ be two adjoining triangular faces. The edges of these faces form a quadrangle $acbd$ with the diagonal (a, b). When this diagonal is replaced by (c, d), one obtains a new maximal planar graph G' with the same vertices and the same number of edges and faces. We say that G' has been obtained from G by a **diagonal transformation** or a **diagonal flip**. This is possible only if (c, d) is not an edge in G already.

Additional details may be found in [4, 6].

2.2 Reverse Search Essentials

The reverse search method is developed by Avis and Fukuda [2]. It was used to generate all rooted triangulations (without degree constraint) by Avis [1] and dually to generate all 3-polytopes by Deza et al [3]. It 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 rooted MPG4 with given n. U is the edge set of adjacent vertices in T. Two rooted MPG4 are adjacent if they differ by one edge or two edges. The method works by finding a spanning tree in the graph H. To do this we first fix some given rooted MPG4 with the required parameter n. Call the vertex in H corresponding to this triangulation the **target** E_n^*. We then define a **local search** function L, that given any vertex E_n 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

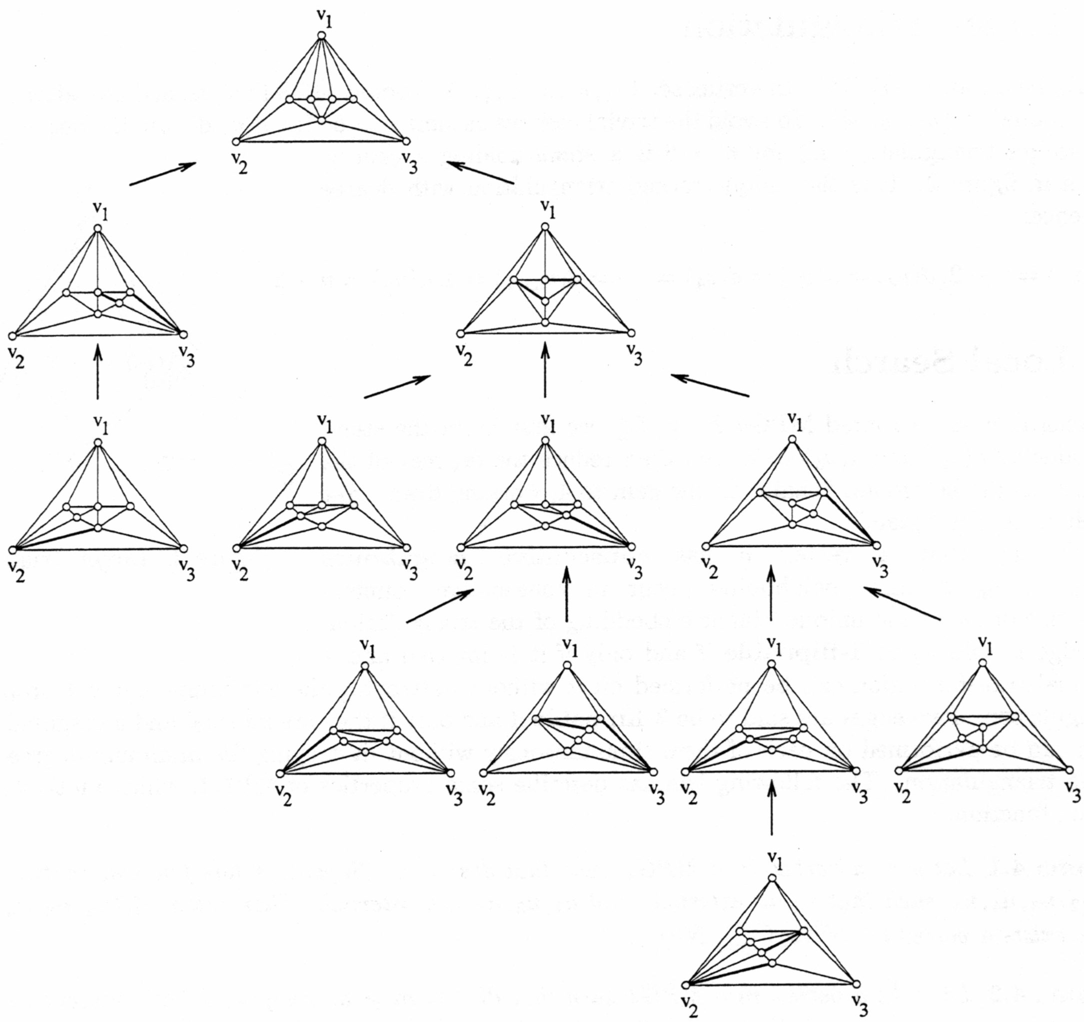

Figure 1: Reverse search tree of rooted MPG4 for $n = 8$

other words, $(E_n, L(E_n))$ is an edge in U and $L^k(E_n) = E_n^*$ for some finite integer k. The path generated consists of a sequence of rooted MPG4 (for given n) that differ in one edge or two edges and ends with the target MPG4. The set of all such paths form a spanning tree called the **reverse search tree** in H. Figure 1 shows the reverse search tree of rooted MPG4 for $n = 8$. The local search function is described in section 4. The arrows show the traces of applications of the local search function. The bold-faced edges are those on which diagonal flips are performed.

The reverse search procedure is essentially a depth first search. It is initiated at the target E_n^* and traverses the reverse search tree of H by *reversing* the local search function. To do this we generate all neighbours in H of any given vertex E_n in T, in some given order. This is done by an **adjacency oracle**. Using the adjacency oracle at E_n^* we consider neighbours of E_n^* until we find a neighbour E_n such that $L(E_n) = E_n^*$. We then replace E_n^* by E_n and use the adjacency oracle to find (if possible) a neighbour E_n' of E_n such that $L(E_n') = E_n$. If such a vertex exists we move to E_n' and continue. If we reach a node E_n for which no such neighbour exists, we backtrack by computing the parent $E_n^0 = L(E_n)$ of E_n. We then continue from E_n^0 using the adjacency oracle to give the next neighbour of E_n^0 in the order after E_n.

Hence it is necessary to specify:

- the target MPG4,
- the local search function, and
- the adjacency oracle.

3 Target Triangulation

Let E_n be a rooted MPG4 with vertex set $\{v_1, v_2, ..., v_n\}$. All rooted MPG4 mentioned hereafter are assumed to be rooted at $\{v_1, v_2, v_3\}$. To avoid the trivial case we assume $n > 6$. Let $d(v)$ denote the degree of vertex v. The target triangulation E_n^* for $n > 6$ is a *stand* holding a *gem* as shown in figure 2. It is the unique rooted triangulation with degree sequence:

$$d(v_1) = n - 2, d(v_2) = d(v_3) = d(v_4) = \cdots = d(v_{n-1}) = 4, d(v_n) = n - 2$$

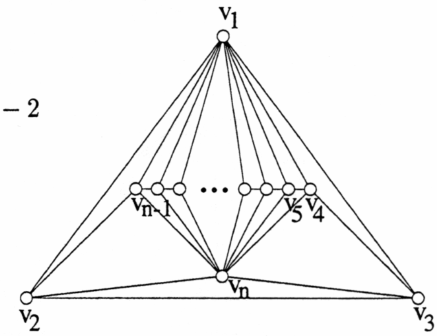

Figure 2: Target Triangulation

4 Local Search

In general, given a rooted MPG4 $E_n \neq E_n^*$, we first build the stand by reducing $d(v_2)$ and $d(v_3)$ to 4, and then reduce the degrees of the vertices in the horizontal "chain" in the gem to 4, working from right to left along the "chain".

We say that a vertex v has **consecutive neighbours** $u_1, u_2, \ldots, u_m$ if these neighbours occur in consecutive counter-clockwise order in the unique planar embedding of the triangulation. An edge is said to be **1-flippable** if and only if it is internal and a diagonal transformation can be performed on it without destroying the minimum-degree-4 property of the triangulation. Two edges are said to be **2-flippable** if and only if they are internal and a diagonal transformation can be performed on each of them simultaneously without destroying the minimum-degree-4 property of the triangulation. The following lemmas describe some properties of MPG4, which motivate our Local Search function.

Lemma 4.1 *Let s be a vertex in a MPG4 such that $d(s) \geq 5$. Suppose s has five consecutive neighbours u_1, u_2, u_3, u_4, u_5 such that u_1 is external, and u_2, u_3, u_4 are internal. Then either $d(\{u_2, u_3, u_4\}) \geq 5$, or there exists a vertex $t \in N^2(s) \bigcap_{j=1}^{5} N(u_j)$.*

Lemma 4.2 *Let s be a vertex in a MPG4 such that $d(s) = m \geq 5$. Suppose s has consecutive neighbours $u_1, u_2, u_3, \ldots, u_{m-1}, u_m$ such that u_1 is external, and $u_2, u_3, \ldots, u_{m-1}$ are internal. If $d(u_2) = d(u_3) = \cdots = d(u_{m-2}) = 4$ and $d(u_{m-1}) \geq 5$, then either there exists a vertex $t \in N^2(s) \bigcap_{j=1}^{m-1} N(u_j)$ such that $d(t) \geq 5$, or (u_1, u_{m-1}) is 1-flippable. Further, (s, u_{m-1}) is 1-flippable.*

Lemma 4.3 *Let s be a vertex in a MPG4 such that $d(s) \geq 5$. Suppose s has four consecutive neighbours u_1, u_2, u_3, u_4 such that u_2, u_3 are internal, and $d(u_2) \geq 5$. Then either (s, u_2) is 1-flippable, or (s, u_3) is 1-flippable. Further, if (s, u_2) is not 1-flippable, then $d(u_1) \geq 5$.*

Lemma 4.4 *Let s be a vertex in a MPG4 such that $d(s) \geq 5$. Suppose s has four consecutive neighbours u_1, u_2, u_3, u_4 such that u_1 is external, u_2, u_3, u_4 are internal, and $d(u_2) = d(u_3) = 4$. Let $t \neq s$ be the common neighbour of u_1, u_2, u_3, u_4, and $d(t) \geq 5$. Then (s, u_2) and (t, u_3) are 2-flippable.*

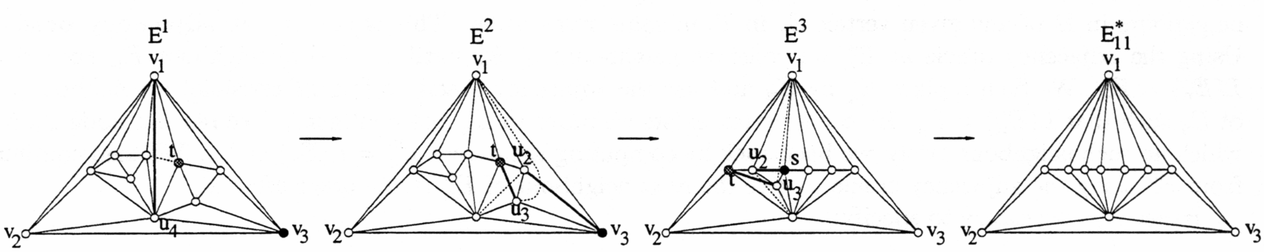

Figure 3: Illustrating the local search function

Before presenting the Local Search function, let us illustrate it with an example. The idea is to compare first, the degrees of vertices v_2 and v_3 to their corresponding degrees in the target, and then the degrees of vertices along the horizontal "chain" in the gem to their corresponding degrees in the target, starting with the vertex adjacent to v_1 and v_3, and examine the neighbours of v_1 in clockwise order.

Figure 3 shows an example with $n = 11$. E^1 has $d(v_3) = 5$. Since $d(t) < 5$, we apply Lemma 4.2 and flip (v_1, u_4) to get E^2. In E^2, $d(v_3)$ is still > 4. We can apply Lemmas 4.2 and 4.4, flip (v_3, u_2) and (t, u_3) to get E^3. We see that $d(s) = 5$ in E^3. Again we can apply Lemmas 4.2 and 4.4, flip (s, u_2) and (t, u_3) and then reach our target E^*_{11}.

Figure 4 shows the pseudo code for the Local Search function `LocalSearch`. In our notation, `deg(v)` denotes $d(v)$, and `v[i]`, `u[j]` denote vertices v_i and u_j respectively. Hereafter, "application" of the function implies the diagonal transformation follows on the returned edge(s).

The next theorem shows that repeated application of the function leads any rooted MPG4 to the target.

Theorem 4.1

i. Edges returned by `LocalSearch` *are either 1-flippable or 2-flippable.*

ii. Successive application of `LocalSearch` *transforms any rooted MPG4* $E_n (n \geq 6$ *and* $E_n \neq E^*_n)$ *to the target MPG4* E^*_n.

Sketch of proof: For part *i*, we use the lemmas. Whenever two edges are returned, we apply Lemmas 4.2 and 4.4. Whenever (v_i, u_j), (v_i, u_{j+1}), (s, u_j) or (s, u_{j+1}) is returned, we apply Lemma 4.3. Whenever (v_i, u_{m-1}) or (u_1, u_{m-1}) is returned, we apply Lemma 4.2. In all cases, s in the lemmas will be the vertex v_i or s in `LocalSearch`. By the lemmas, the returned edges are either 1-flippable or 2-flippable. Hence, we have shown that application of `LocalSearch` on any rooted MPG4 always results in another rooted MPG4 (for given n).

For part *ii*: for $2 \leq i \leq 3$, the algorithm reduces $d(v_i)$ by one, by returning a 1-flippable edge incident to v_i; or it returns some edge, flipping which leads to another MPG4, on which subsequent application of `LocalSearch` returns a 1-flippable edge incident to v_i, hence reducing $d(v_i)$ by one. We need to show that the process of reducing $d(v_3)$ does not increase $d(v_2)$. This can be verified by going through all the cases.

When $d(v_2)$ and $d(v_3)$ reach 4, the algorithm works on the vertices that are internal neighbours of v_1 in clockwise order. We have to show that, when $s=$ the vertex adjacent to v_1 and v_3, and $lasts = v_3$, the process of reducing $d(s)$ does not increase $d(lasts)$ or $d(v_2)$. Again this can be verified by going through all the cases.

By the structure of the target triangulation we defined, the above argument applies to subsequent s's for $lasts \neq v_3$ as well, since we can assume that previous $lasts$'s are erased from the graph and this part will not be affected by the algorithm anymore. The process stops once the degrees of the vertices match their counterparts in the target, and so the algorithm is finite. □

5 Adjacency Oracle

Suppose E_n has its edges indexed from 1 to $3n - 6$, and let e_1, e_2, e_3 be the 3 external edges. The edges are stored in some ordered edge list $E_n = \{e_1, e_2, \ldots, e_{3n-6}\}$.

Suppose a single diagonal flip is performed on some internal edge e. The new edge e' is well defined and need not be specified explicitly. The operation is: $\text{flip}_1(E_n, e) = E_n - e + e'$.

We know that not only a single diagonal flip, but two simultaneous diagonal flips can also define adjacency. Suppose a double diagonal flip is performed. The two edges flipped must be the two pairs of opposite sides of some quadrangle in the triangulation. Each quadrangle is defined by one internal edge in the triangulation. Hence for each internal edge in the triangulation, we also consider two pairs of edges. We can define the adjacency oracle which gives all adjacent rooted MPG4 to a given rooted MPG4 E_n as follows:

```
LocalSearch(E,n) // returns one(or two) flippable edge(s) or (-1) if E==target

// reduce deg(v[2]) and deg(v[3]) to 4
i=2; while (deg(v[i])==4  &&  i<=3) i++;
if (i<=3){ Let u[1],u[2],...,u[m] be the m consecutive neighbours of v[i].
           u[1] is external, u[2],...,u[m-1] are internal.  // deg(v[i])==m>=5
           if (deg(u[j])>=5 for some minimal j where 2<=j<(m-1))
             { if (u[j-1],u[j+1]) is not an edge return((v[i],u[j]));
               else return((v[i],u[j+1]));
             }
           if (deg(u[j])<5 for all 2<=j<(m-1)  &&  deg(u[m-1])>=5)
             { case (i) of
               2: return((v[i],u[m-1]));
                  // cannot return (v[i],u[m-1]) if i==3, otherwise deg(v[2])
                  // will go up
               3: Let t!=v[i] be the common neighbour of u[1],u[2],...,
                  u[m-1].
                  if (deg(t)<5) return((u[1],u[m-1]));
                  else return((v[i],u[2]) and (t,u[3]));
             }
           if (deg(u[j])<5 for all 2<=j<=(m-1))
             { Let t!=v[i] be the common neighbour of u[1],u[2],...,u[m].
               return((v[i],u[2]) and (t,u[3]));
             }
        }
// build the gem
Let s be the vertex adjacent to v[1] and v[3], lasts be v[3].
while (deg(s)==4  &&  s!=the vertex adjacent to v[1] and v[2])
     { lasts=s; s=the neighbour of v[1] previous to s in counter-clock-
       wise order;
     }
if (s==the vertex adjacent to v[1] and v[2]) return (-1); // E==target

Let u[1]=v[1], u[2],...,u[m]=lasts be the m consecutive neighbours
of s.  // deg(s)==m>=5
if (deg(u[j])>=5 for some minimal j where 2<=j<(m-1))
  { if (u[j-1],u[j+1]) is not an edge return((s,u[j]));
    else return((s,u[j+1]));
  }
if (deg(u[j])<5 for all 2<=j<(m-1)  &&  deg(u[m-1])>=5)
  { // cannot return (s,u[m-1]), otherwise deg(lasts) will go up
    Let t!=s be the common neighbour of u[1],u[2],...,u[m-1].
    if (deg(t)<5) return((u[1],u[m-1]));
    else return((s,u[2]) and (t,u[3]));
  }
if (deg(u[j])<5 for all 2<=j<=(m-1))
  { Let t!=s be the common neighbour of u[1],u[2],...,u[m].
    return((s,u[2]) and (t,u[3]));
  }
```

Figure 4: Pseudo code for the Local Search function

For $j = 4$ to $(3n-6)$, let $e_j = (a, b)$ bound $\triangle abc$ and $\triangle abd$. Define

$$\mathrm{Adj}_1(E_n, j) = \begin{cases} \mathrm{flip}_1(E_n, e_j) \text{ if } e_j \text{ is 1-flippable} \\ \emptyset \text{ otherwise} \end{cases}$$

$$\mathrm{Adj}_2(E_n, j) = \begin{cases} \mathrm{flip}_1(E_n, (a,c)) \text{ and } \mathrm{flip}_1(E_n, (b,d)) \text{ if } (a,c) \text{ and } (b,d) \text{ are 2-flippable} \\ \emptyset \text{ otherwise} \end{cases}$$

$$\mathrm{Adj}_3(E_n, j) = \begin{cases} \mathrm{flip}_1(E_n, (a,d)) \text{ and } \mathrm{flip}_1(E_n, (b,c)) \text{ if } (a,d) \text{ and } (b,c) \text{ are 2-flippable} \\ \emptyset \text{ otherwise} \end{cases}$$

n	#rooted MPG4	#rooted MPG5	time taken/sec.	#rooted MPG	time taken/sec.
4	-	-	-	1	0
5	-	-	-	3	.002
6	1	-	0	13	.003
7	3	-	0	68	.010
8	12	-	.007	399	.060
9	59	-	.037	2,530	.506
10	313	-	.255	16,965	4.355
11	1,713	-	1.850	118,668	37.762
12	9,559	1	13.224	857,956	330.423
13	54,189	0	92.120	6,369,883	2911.029
14	311,460	6	638.280	?	?
15	1,812,281	13	4459.693	?	?
16	10,661,303	55	30424.496	?	?
17	63,336,873	189	207669.911	?	?

Table 1: Results available

6 Results

For brevity the code for the reverse search algorithm is omitted. It is essentially the same as the standard reverse search procedure given in [2]. We have implemented the algorithm in C using the same data structure adopted from the program in [1]. Complexity analysis is straightforward as in [1]: the algorithm uses $O(n)$ space and $O(n^2 \cdot f(n))$ time where $f(n)$ is the number of rooted MPG4 on n points. It was tested on a DEC3000/500 Alpha for $6 \leq n \leq 17$. For $6 \leq n \leq 13$, the total number of rooted MPG4 were verified with the program in [1]. The largest problem solved with $n = 17$ and $f(n) = 63,336,873$ took 3461 minutes. We have also added a constraint to our program to produce rooted MPG5. These results and timings are shown in table 1.

7 Conclusion and Future Work

We have shown that the graph of rooted MPG4 (given n) is connected under single and double diagonal transformations. Hence we designed and implemented an efficient algorithm for generating non-isomorphic rooted MPG4 using reverse search. It is possible to parallelize our program for better run time and we may want to do this to obtain results for larger n. Hurtado and Noy [5] showed the graph of rooted triangulations is hamiltonian. It would be interesting to see if the same is true for the graph of rooted MPG4. Finally, we found that the graph of rooted MPG5 (given n) is not connected under single and double diagonal transformations alone. However, it might be connected under single, double and some triple diagonal transformations (in fact the graph of rooted MPG5 for $n = 14$ verifies this) and this deserves further investigation.

References

[1] D. Avis. Generating rooted triangulations without repetitions. Technical Report SOCS-94.2, McGill University, 1994. To appear in *Algorithmica.*

[2] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 6:21–46, 1996.

[3] A. Deza, K. Fukuda, and V. Rosta. Wagner's theorem and combinatorial enumeration of 3-polytopes. Combinatorial Geometry B-271, Department of Information Sciences, Tokyo Institute of Technology, October 1993.

[4] S. Even. *Graph Algorithms.* Computer Science Press, 1979.

[5] F. Hurtado and M. Noy. The graph of triangulations of a convex polygon. Technical Report MA2-IR-94-13, Departament de Matemàtica Aplicada II, Universitat Politècnica de Catalunya, May 1994.

[6] O. Ore. *The Four-Color Problem.* Academic Press, 1967.