Some observations related to the 4CT

Van den Driessche Willy

In this document I describe some observations I made when tackling the 4CT. I have not encountered them in any previous work. The most surprising fact is that we can generate all maximal planar graphs with just 3 rules. Rule 1 keeps the number of 4-colourings constant, rule 2 at most halves it and rule 3 at most divides it by 4.

1 Maximal planar graphs

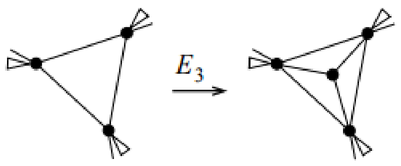
I approached the 4CT like many other people have tackled it. I came to the conclusion that it was sufficient to concentrate on *maximal* planar graphs (MPG); graphs that lose their planarity if you add another edge to them.

2 Three rules

Another thing I wanted to do is approach these MPG in a generative way. The idea was to "order" them by number of vertices so that we could apply some kind of induction rule on them.

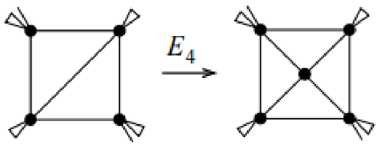
It is sufficient to start with K4. From there on, we generate MPGs. To get from an MPG with n nodes to an MPG with n+1 nodes, I found 3 rules. (I found these rules later in at least 3 papers giving me confidence that this was correct.)

Rule1 is simple : Embed the planar graph. Embed a new vertex in the middle of any face. Connect the vertex to the 3 boundary vertices of the previous face. This can be visualized as follows :



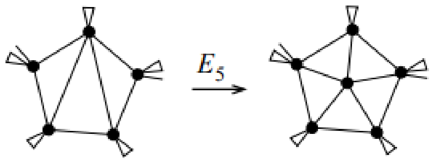
.[1]

Rule2 is equally simple : Add a new vertex in the middle of any edge. Connect the new vertex to the vertices on the faces.



[1] I stole these images without permission from the paper by Gunnar Brinkmann and Brendan D. McKay. The special marks at the vertices are used there to avoid rules that produce isomorphic graphs. We ignoree them here since we want to be sure not to miss any rule production.

Rule3 came much later (after I saw that my calculations had less solutions than standard implementations). It involves a pentagon and can be visualized as follows :



These 3 rules have been used by others. The Plantri program uses them to generate MPGs in the blink of an eye.

I rendered these MPGs, hoping to see some patterns in them.

I quickly put this aside because I realized a lot of people had already looked at these graphs.

3 The transformations are key

Then I realized that I had to see the *transformations* as the key of my approach.

Instead of looking at an MPG, I decided to look at the rules and to see what invariants they carried with them.

The "hoped for" invariant would of course be that each rule moves from a 4-colourable MPG to another 4-colourable MPG.

3.1 Rule 1

It was clear that Rule1 did not change the way you could colour the graph with 4 colours. If the original graph could be coloured with 4 colours in n ways, then after applying rule 1 to it, we would have a new graph with one more vertex but which could also be coloured in the *same* n ways. The colour of the new face center is completely determined by the original corners of the triangular face.

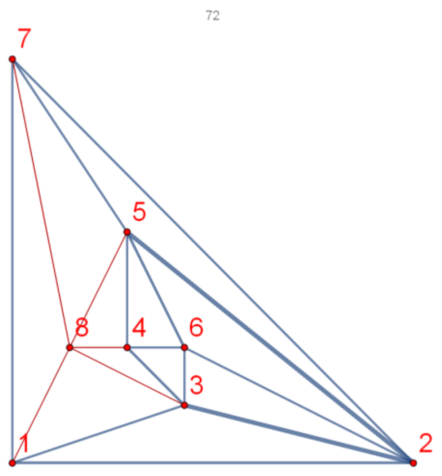
3.2 Rule 2

3.2.1 The idea

The goal was to have the same approach for rule 2. If we could start with a 4-colourable graph with n nodes that could be coloured in k ways, then we hoped that applying rule 2 would always lead us to a graph with n+1 nodes that could be 4-coloured in also *at least* k ways. Luckily for me, Mathematica has a built-in `ChromaticPolynomial` method. If we feed 4 into that polynomial, we can see the number of possible ways to colour the graph with 4 colours. (I actually programmed this myself too and reached the same results but an order of magnitude slower ;-))

3.2.2 They do go down

Unfortunately, I quickly found that this doesn't hold. The following picture shows a graph to which rule 2 is being applied :



In this graph, vertex 8 has been added on an edge between vertex 1 and 5. The graph *without* node 8 can be 4-coloured in **96** ways, while this graph can be 4-coloured in only 72 ways. So while there is an additional vertext, the 3 additional edges make it so that the number drops.

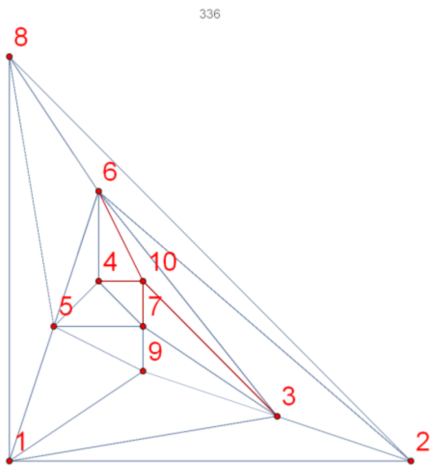
Ok. That was a disappointment.

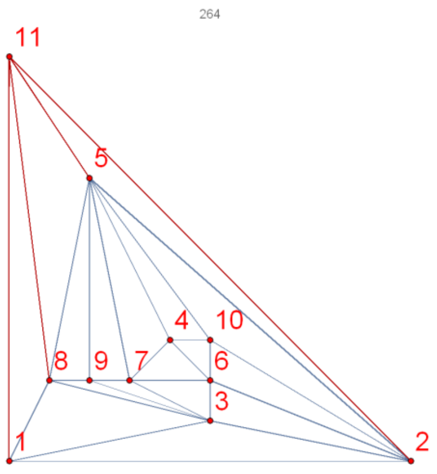
3.2.3 They go down in chains

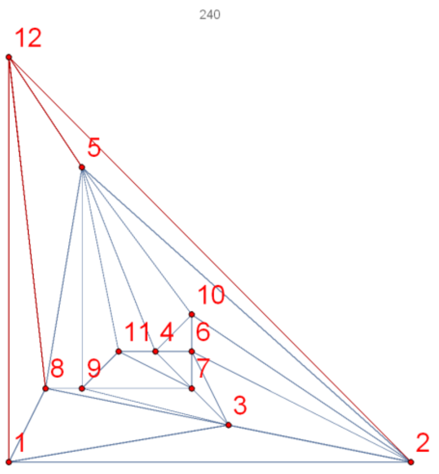
But I thought. Ok, the number goes down, but it only goes down once and afterwards it goes back up again. Unfortunately, I found some chains of graphs that have successive lower numbers.

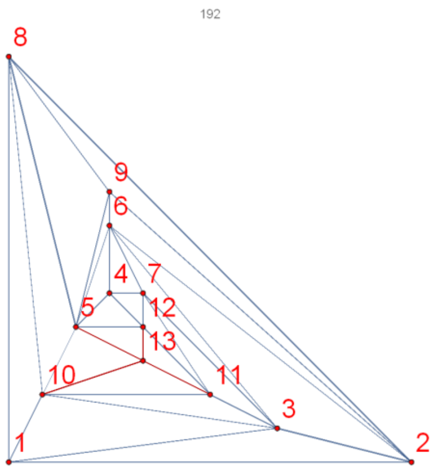
The graphs in the follwing pictures have succesively more vertices. But they go from 336 possibilities down to 144. In each step we add a node (you can see which necause the vertex numbers increase).

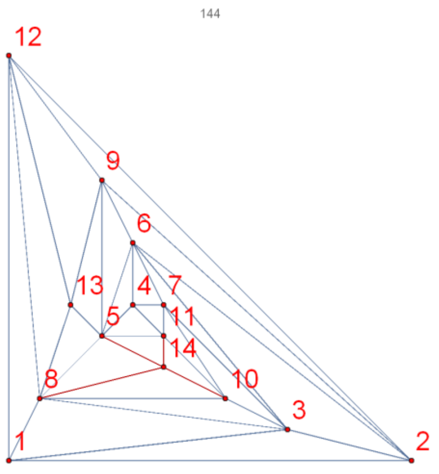
This experiment gave the hint that these chains could be as long as possible if we wanted.


336
8
6
4
10
5
7
9
3
1
2


264
11
5
4
10
8
9
7
6
3
1
2

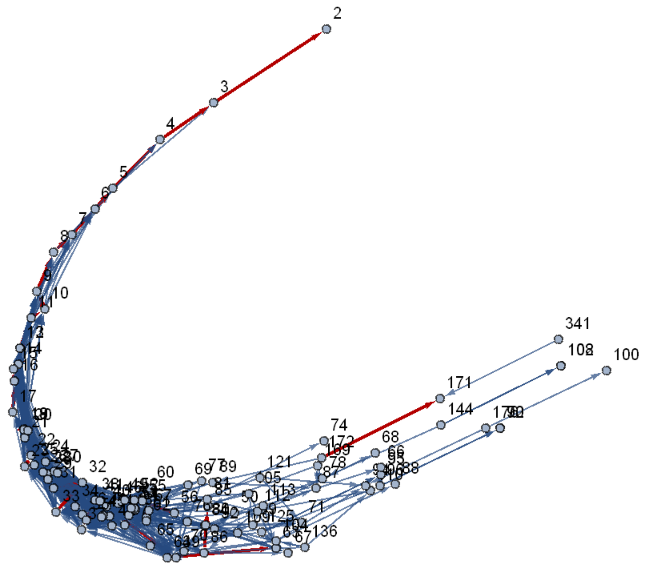

240
12
5
10
11
4
6
8
9
7
3
1
2


192
8
9
6
4
7
5
12
13
10
11
3
1
2


144
12
9
6
13
4
7
5
11
14
8
10
3
1
2

3.2.4 Visualizing the transitions

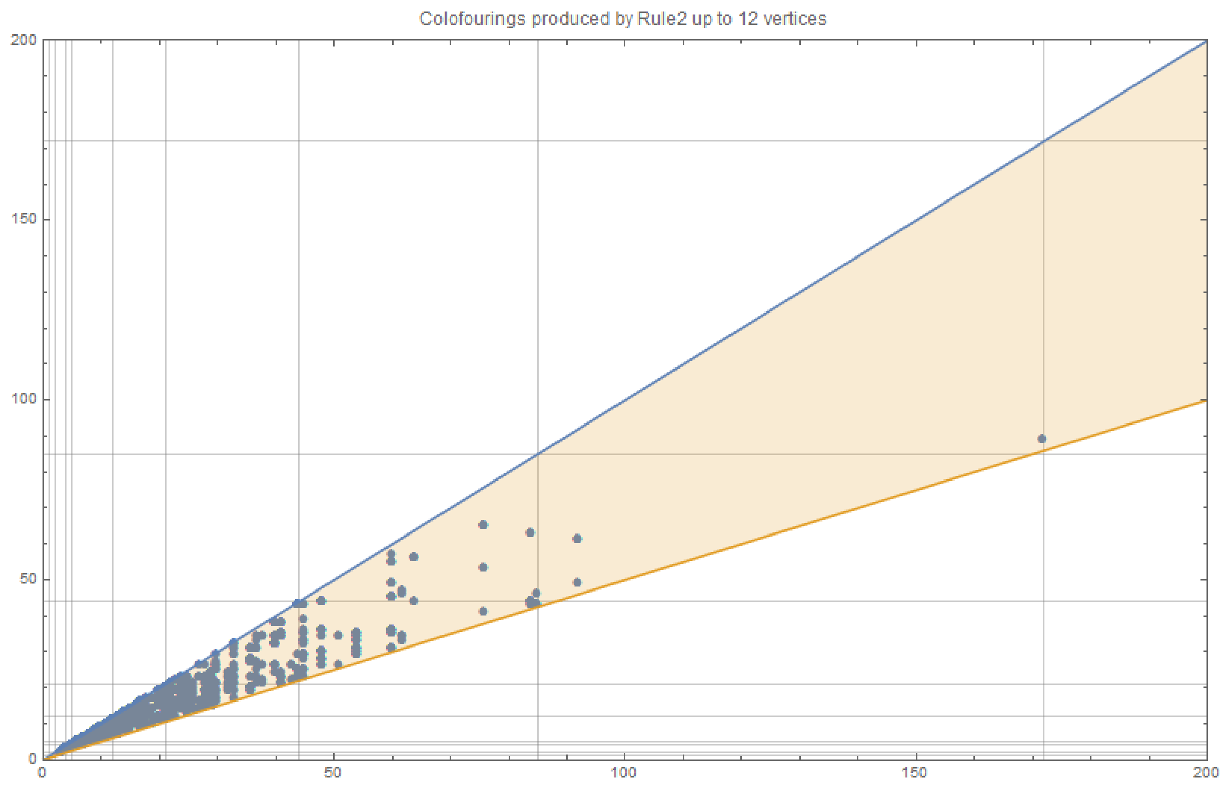
My first attempt was to visualize the "color transitions" for rule2 through a graph. If the colors drop from 48 to 24 then I would add vertices 48 and 24 and connect them through an edge. I would then ask Mathematica to draw that graph to get an idea of these transitions. This gave the following solution :



The above graph contains the number of 4 colourings divided by 24. This gives easier numbers. Let's call this number the *colofouring.*

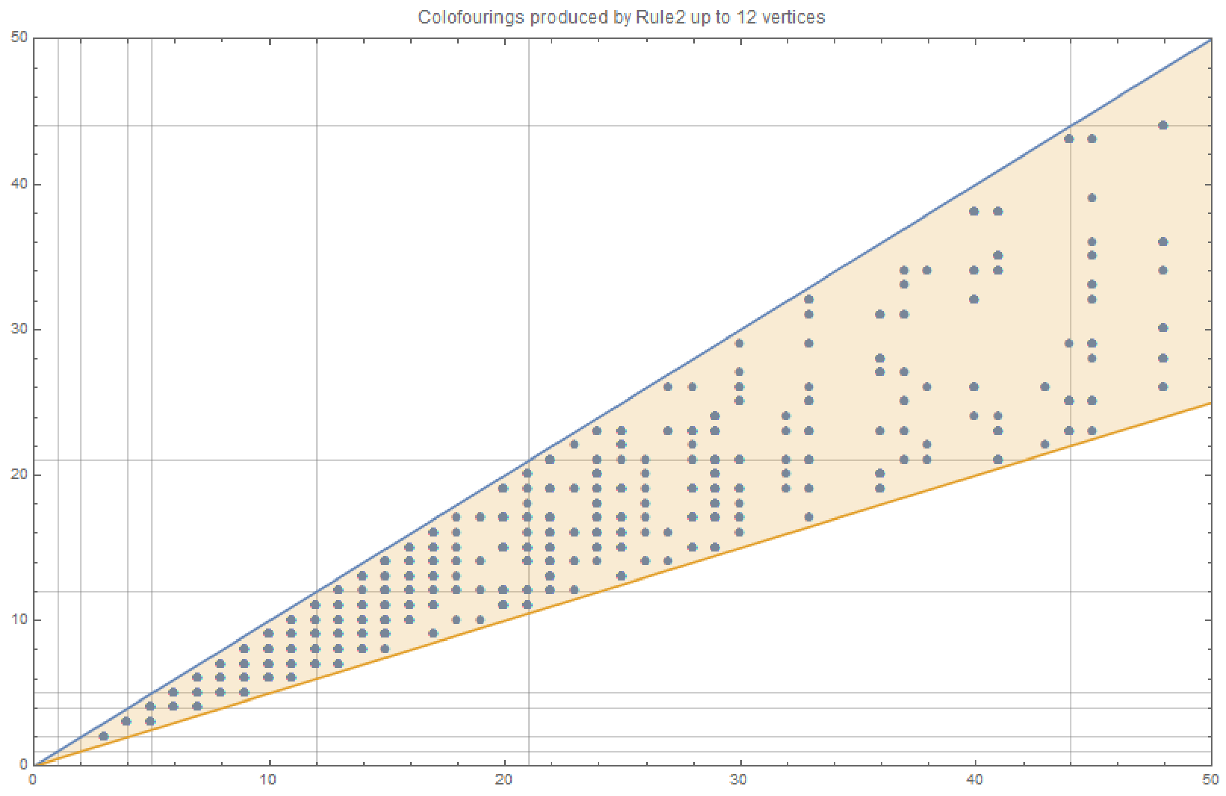
This was suboptimal but it suggested something. Apparently colofour 2 is only reached from colofour 3; 3 is only reached through 4 and 5.

Then, by accident actually, I plotted these "edges" as points in a normal carthesian graph. This produced the following graph :



Let's be very clear about what is in this graph. A dot (x,y) is in the graph if there is a graph g with N nodes and colofouring x is transformed into a graph h with n+1 nodes an colofouring y through a rule2 transition. In addition we ask that y<=x. There are more transitions than the one shown. But we only show the when the colofouring drops (or stays the same). (There is a graph with *all* numbers a bit further in the document)

When I saw this grah I was *completely* surprised. If we zoom in on the x<=50, we see this:



We notice that *most* (x,y) values are there. However, we also notice that apparently a rule 2 will never go lower than half of the previous colofouring. Actually, it won't even drop to half the previous. It will always be one more.

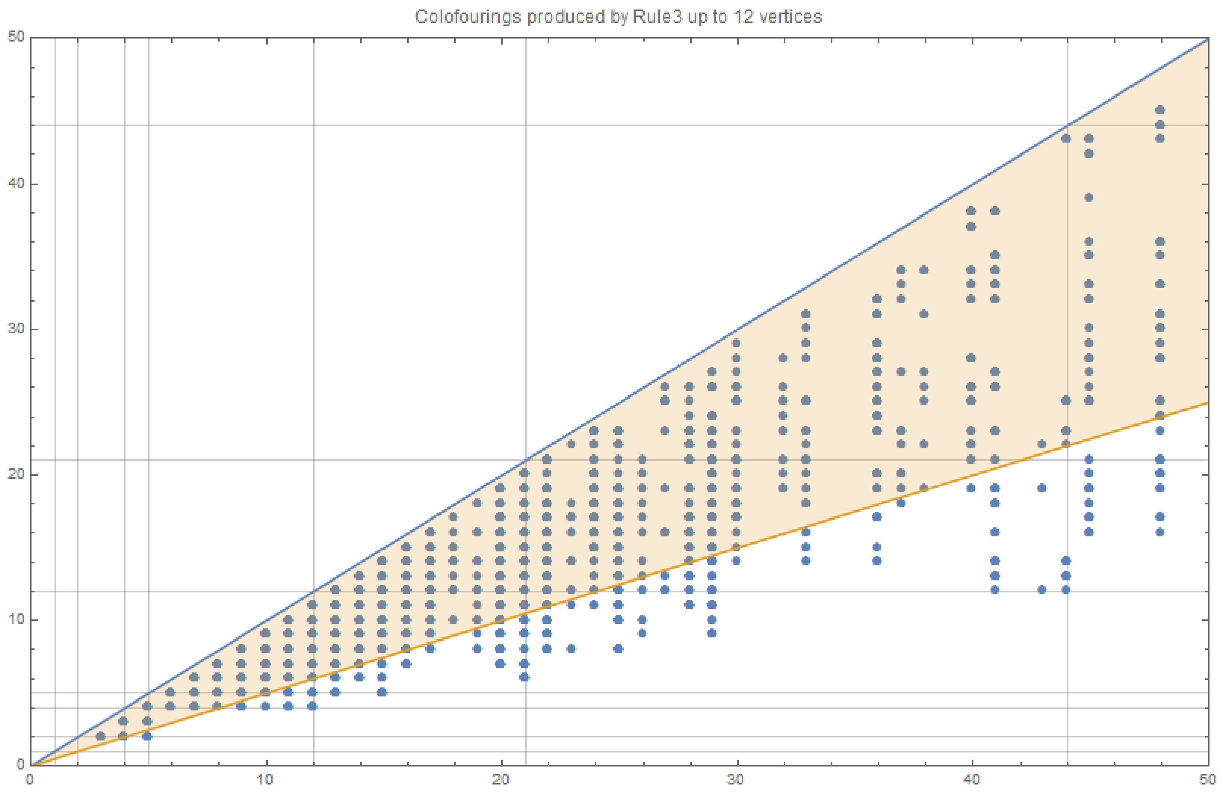
Now this is a property about rule 2 that I have found nowhere. If we could prove this property then we would have solved 66% of the 4CT (counting rule 1 as the easiest 33%)

Conjecture 1. *A rule 2 transition from F to G will never bring the colofouring down to less than or equal half of the original colofouring.*

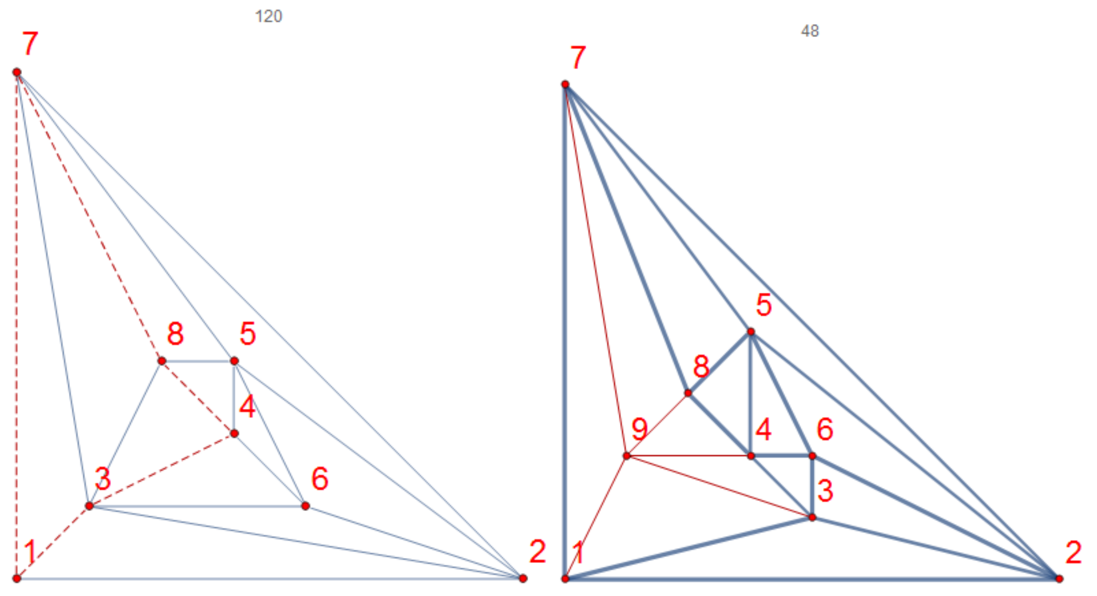
3.3 Rule 3

3.3.1 The idea : same as rule2

With the image of rule2, I quickly wrote a program to collect rule3 transitions. I expected rule 3 to behave exactly like rule 2. Unfortunately, I got the following picture:

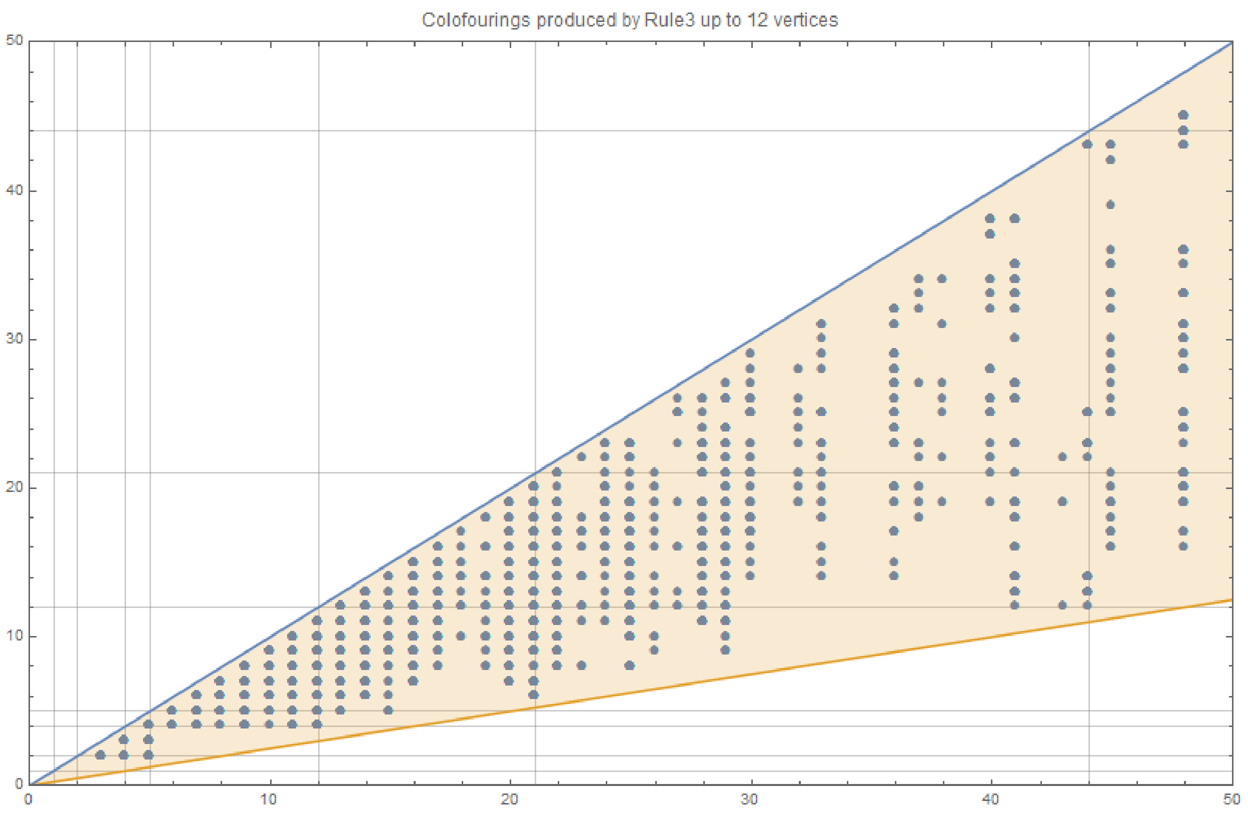


You can see that the blue dots below the orange line don't obey the rule2 conjecture. The lowest such colofouring transition happens at 5,2. The graphs that go with this transition are shown below (vertex 9 will be added in the pentagon 7,8,4,3,1)



3.3.2 Second idea : other rule, other law

But then I realized that rule 3 and rule2 are quite different. So there is no reason for rule3 transitions to obey *exactly* the *same* law as rule2. This leads me to the idea that the law for rule3 is *similar* (but not the same) to the law of rule 2. In a picture this gives us:



Conjecture 2. *A rule 3 transition from F to G will never bring the colofouring down to less than or equal one fourth of the original colofouring.*

Again, If we would be able to prove this, we would have proven another 33% of the 4CT.

From the picture of rule3 the 1/4th rule is not all that obvious. With the limited data this law looks ok, but a more elaborate calculation this law might quickly break down.

4 Observation about maximum x for count of vertex

When we calculate the x,y for the colofouring transition graph, we do so for increasing vertex counts. The reason is quite obviously that up until now I need a generation with n vertices before I can calculate a generation with *n+1* vertices. (I *could* use Plantri to fast forward to *m* nodes but I would still need to calculate the transitions to *m+1* nodes)

I noticed that for vertex count n, there was huge gap between the maximum x and the other "x's". So I got interested in *knowing* the maximum x for a certain vertex count.

This got me the table { 1, 4, 5, 12, 21, 44, 85, 172, 341, 684, 1365, … } Thanks to OEIS I found that these are the so-called A155944 sequence of "Jacobsthal numbers A001045, every second term incremented by 1.". I have no idea how to explain this but it leads to another conjecture.[2]

Conjecture 3. *The colofouring transition (x,y) with maximum x for a given vertex count is related to the Jacobsthal numbers.*

Proving this conjecture might provide a "horizontal" (because it's about x) upper bound that might be useful for "framing" problems for the other conjectures.

5 Observation about set of colofouring transitions for a given vertex count.

I made another observation regarding colofouring transitions. Suppose KF(n) is the set of all (x,y) so that (x,y) is a colofouring transition that happens between a graph with *n* nodes to a graph with *n+1* nodes by some of the 3 rules.

To my surprise I noticed that KF(n) is always a *proper subset* of KF(n+1).

Conjecture 4. *The colofouring transitions for vertex counts n are a proper subset of the colofouring transitions of vertex counts n+1.*

For this conjecture I have the feeling that there is some obvious observation that I am missing. It has the flavor of being trivial. Yet I don't see the reason now.[3]

Here are some sets related to rule 2 to make this observation more clear: (In row *n+1* we color the items from row *n* to show that they indeed occur – we underline drops)

Vertex count	**Unique colofouring transitions for rule 2 (dropping and non-dropping)**
7	{1,1},{1,5},{4,5}

[2] In fact there is a paper that links Jacobstal numbers to generalized Petersen graphs (http://eurocomb2015.b.uib.no/files/2015/08/endm1943.pdf) so the link might not be as far-fetched.
[3] The recurrence of the Jacobsthal numbers

Vertex count	**Unique colofouring transitions for rule 2 (dropping and non-dropping)**
8	{1, 1}, {1, 4}, {1, 5}, {1, 12}, {4, 3}, {4, 4}, {4, 5}, {5, 3}, {5, 12}
9	{1,1},{1,4},{1,5},{1,12},{4,3},{4,4},{4,5},{4,9},{4,16},{5,3},{5,5},{5,6},{5,9},{5,12}
10	{1,1},{1,4},{1,5},{1,12},{1,21},{3,2},{3,6},{3,9},{4,3},{4,4},{4,5},{4,9},{4,11},{4,16}, {4,20},{5,3},{5,5},{5,6},{5,7},{5,9},{5,12},{5,20},{12,7},{12,9},{12,11},{12,12},{12,21}
11	{1, 1}, {1, 4}, {1, 5}, {1, 12}, {1, 21}, {1, 44}, {2, 5}, {2, 6}, {2, 8}, {3, 2}, {3, 3}, {3, 5}, {3, 6}, {3, 9}, {3, 12}, {3, 13}, {3, 14}, {3, 22}, {4, 3}, {4, 4}, {4, 5}, {4, 9}, {4, 11}, {4, 16}, {4, 20}, {4, 25}, {5, 3}, {5, 5}, {5, 6}, {5, 7}, {5, 9}, {5, 12}, {5, 14}, {5, 20}, {5, 25}, {5, 28}, {5, 41}, {6, 5}, {6, 6}, {6, 7}, {6, 13}, {6, 14}, {9, 5}, {9, 7}, {9, 8}, {9, 9}, {9, 10}, {9, 11}, {9, 14}, {9, 22}, {9, 28}, {9, 29}, {12, 7}, {12, 9}, {12, 11}, {12, 12}, {12, 13}, {12, 21}, {12, 41}, {16, 12}, {16, 14}, {16, 16}, {16, 20}, {21, 11}, {21, 14}, {21, 21}, {21, 25}, {21, 44}
12	{1,1},{1,4},{1,5},{1,12},{1,21},{1,44},{1,85},{2,2},{2,5},{2,6},{2,8},{2,9},{2,16},{2,26},{3,2},{3,3},{3,5},{3,6},{3,9},{3,12},{3,13},{3,14},{3,15},{3,22},{3,29},{3,45},{3,54},{4,3},{4,4},{4,5},{4,9},{4,11},{4,16},{4,20},{4,25},{4,48},{5,3},{5,4},{5,5},{5,6},{5,7},{5,8},{5,9},{5,10},{5,12},{5,14},{5,16},{5,17},{5,20},{5,25},{5,28},{5,41},{5,60},{6,4},{6,5},{6,6},{6,7},{6,8},{6,10},{6,11},{6,13},{6,14},{6,16},{6,17},{6,19},{6,21},{7,4},{7,6},{7,7},{7,8},{7,11},{7,13},{7,14},{7,15},{7,16},{7,17},{7,22},{7,29},{8,7},{8,8},{8,10},{9,5},{9,7},{9,8},{9,9},{9,10},{9,11},{9,14},{9,22},{9,23},{9,27},{9,28},{9,29},{9,36},{11,6},{11,10},{11,11},{11,14},{11,17},{11,22},{11,25},{11,41},{11,45},{11,54},{12,7},{12,9},{12,11},{12,12},{12,13},{12,21},{12,41},{12,48},{12,60},{13,7},{13,9},{13,11},{13,13},{13,16},{13,17},{13,22},{14,9},{14,10},{14,11},{14,29},{14,40},{16,10},{16,12},{16,14},{16,16},{16,20},{16,22},{16,36},{20,11},{20,12},{20,15},{20,17},{20,20},{20,24},{20,25},{20,29},{20,36},{20,48},{21,11},{21,14},{21,21},{21,25},{21,44},{28,17},{28,29},{28,41},{44,25},{44,85}
13	Are currently calculating…

A similar table holds for colofouring due to rule 3

Vertex count	Unique colofouring transitions for rule 3 (dropping and non-dropping)
8	{{1,1},{1,4},{1,5},{4,3},{4,4},{4,5},{5,5}}
9	{{1,1},{1,3},{1,4},{1,5},{4,3},{4,4},{4,5},{5,2},{5,3},{5,5},{5,6},{5,9},{5,12}}
10	{{1,1},{1,3},{1,4},{1,5},{1,6},{1,9},{1,12},{3,2},{3,3},{3,6},{3,9},{4,2},{4,3},{4,4},{4,5},{4,6},{4,9},{5,2},{5,3},{5,5},{5,6},{5,9},{5,12},{5,13},{12,6},{12,12}}
11	{{1,1},{1,3},{1,4},{1,5},{1,6},{1,7},{1,9},{1,11},{1,12},{1,21},{2,2},{2,5},{2,6},{2,8},{3,2},{3,3},{3,4},{3,5},{3,6},{3,8},{3,9},{3,11},{3,13},{3,14},{4,2},{4,3},{4,4},{4,5},{4,6},{4,7},{4,9},{4,10},{4,12},{4,13},{4,14},{4,16},{4,20},{5,2},{5,3},{5,5},{5,6},{5,7},{5,9},{5,11},{5,12},{5,13},{5,20},{5,25},{6,5},{6,6},{6,7},{6,8},{6,13},{6,14},{9,5},{9,6},{9,7},{9,9},{9,11},{9,13},{12,5},{12,6},{12,7},{12,9},{12,11},{12,12},{12,21},{16,13},{16,16},{21,7},{21,21}}
12	{{1,1},{1,3},{1,4},{1,5},{1,6},{1,7},{1,9},{1,11},{1,12},{1,21},{1,25},{1,44},{2,2},{2,5},{2,6},{2,7},{2,8},{2,9},{2,10},{2,16},{3,2},{3,3},{3,4},{3,5},{3,6},{3,7},{3,8},{3,9},{3,10},{3,11},{3,12},{3,1

Vertex count	Unique colofouring transitions for rule 3 (dropping and non-dropping)
	3},{3,14},{3,15},{3,16},{3,17},{3,21},{3,29},{3,33},{4,2},{4,3},{4,4},{4,5},{4,6},{4,7},{4,9},{4,10},{4,11},{4,12},{4,13},{4,14},{4,16},{4,20},{4,22},{4,36},{5,2},{5,3},{5,4},{5,5},{5,6},{5,7},{5,8},{5,9},{5,10},{5,11},{5,12},{5,13},{5,15},{5,16},{5,17},{5,20},{5,22},{5,23},{5,25},{5,28},{5,29},{6,4},{6,5},{6,6},{6,7},{6,8},{6,9},{6,10},{6,11},{6,13},{6,14},{6,16},{7,4},{7,6},{7,7},{7,8},{7,10},{7,13},{7,14},{7,22},{7,29},{8,7},{8,8},{8,10},{9,4},{9,5},{9,6},{9,7},{9,8},{9,9},{9,10},{9,11},{9,13},{9,14},{9,15},{9,16},{9,17},{9,19},{9,22},{9,28},{9,29},{9,33},{11,4},{11,6},{11,7},{11,10},{11,11},{11,13},{11,14},{11,17},{11,22},{11,25},{11,26},{11,41},{11,45},{11,54},{12,4},{12,5},{12,6},{12,7},{12,8},{12,9},{12,11},{12,12},{12,14},{12,21},{12,22},{12,30},{12,48},{12,60},{13,8},{13,9},{13,10},{13,11},{13,13},{13,16},{13,17},{13,22},{14,7},{14,8},{14,9},{14,10},{14,11},{14,13},{14,14},{14,16},{16,8},{16,9},{16,11},{16,12},{16,13},{16,14},{16,16},{16,20},{20,7},{20,8},{20,9},{20,11},{20,12},{20,13},{20,15},{20,16},{20,20},{20,22},{20,24},{20,25},{20,30},{20,36},{20,48},{21,6},{21,7},{21,11},{21,14},{21,21},{21,25},{21,44},{28,16},{28,22},{28,28},{44,14},{44,44}}
13	Are currently calculating…

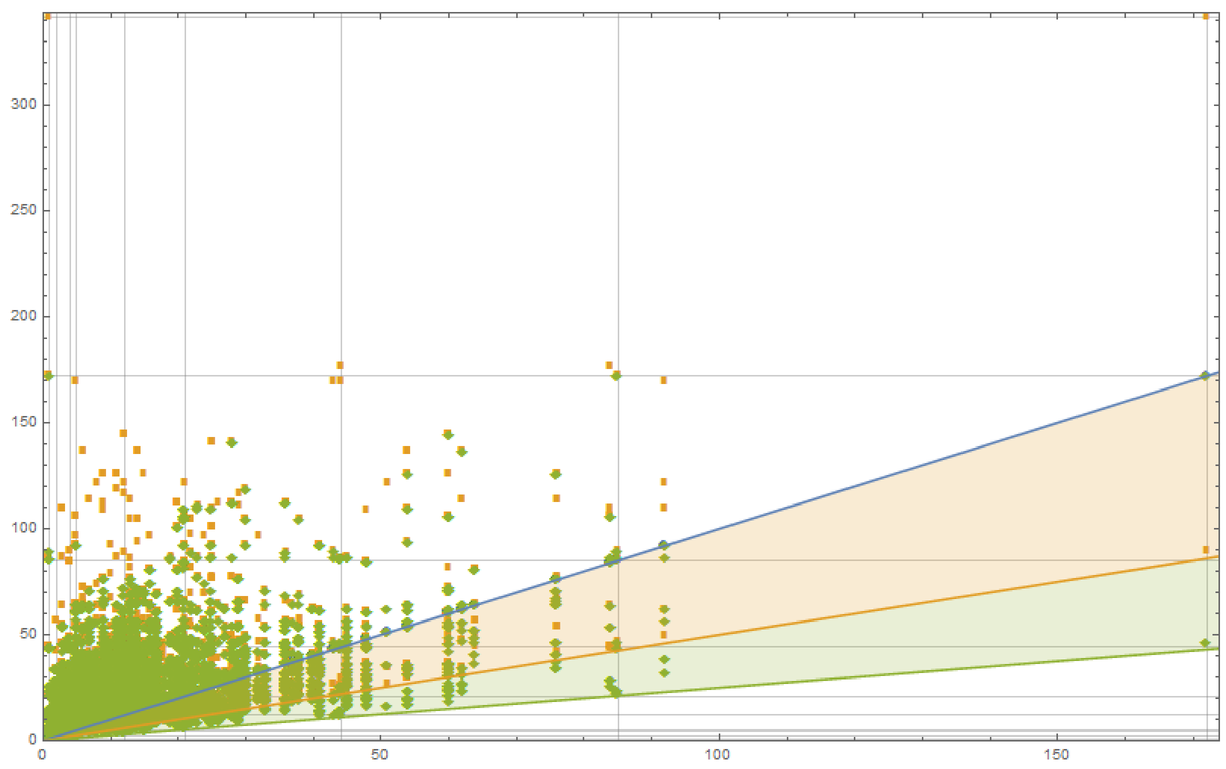
6 Observation about maximum y for count of vertex

If you look at the table from the previous section, then you might notice that for rule 2 the maximum value for y for graphs with vertex count *n* is the maximum value of x for graphs with vertex count *n+1*. For rule3 the maximum value seems to be a Jakobsthal couple (where x and y are equal).
This leads to the following conjecture.

Conjecture 5. The highest colofouring for a graph with n vertices is related to the Jakobsthal numbers.

Proving this conjecture could be easier since this is about the maximum colofouring number of a MPG with n vertices. And somebody must already have solved this ?

The following picture seems to confirm the conjecture :



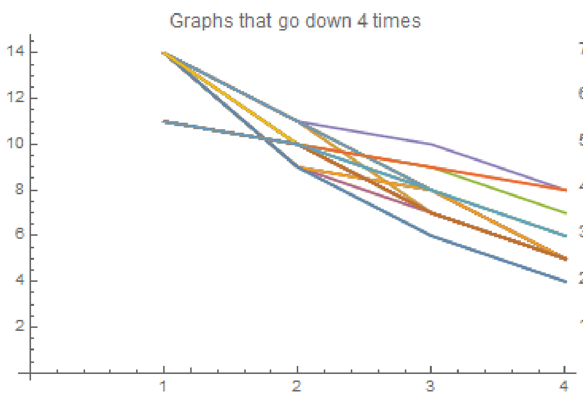
On this graph you can see the colofouring transitions. Blue dots are due to rule1, orange dots due to rule 2, green dots due to rule 3. Since Rule 3 is drawn on top op rule 2 etc you get the false impression that all transitions are rule 3. The gridlines are on the successive Jacobstal numbers. In the upper right corner you can see a dot which seems to confirm the conjecture.

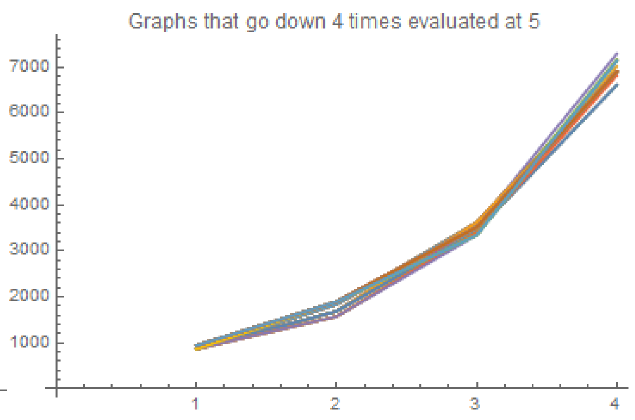
7 Observation about transitions at colofiving

While some of the rule transitions cause a drop in colofouring, it appears to be that colofiving (same as colofour but with five colors to choose from) never drops. Not even for graphs at which colofour drops.

Conjecture 6. *There is no drop in colfiving for rule transitions.*

Here are two graphs to give evidence :





The left graph show some chains for which the colorifour drops 4 times in a row (the x-axis is jst the number of drops in a row, the y-axis show the colofour). The right graph shows these same chains but instead of the colofour we plot the colofive.

8 Observations that border numerology

There are some other observations that are so weak that I barely dare to voice them. However, here it is.

It appears that the linear law for colofouring also applies to five and six colors. Of course, for five and six colors the "y" is always bigger than "x". However, both of them *seem* to follow some kind of (seemingly linear) law. Of course, 12 nodes are not enough to be sure about this.

For that amount of data the difference between a linear and a logarithmic or othe rlaw is too weak.

9 Implementation

The calculation of a MPG is a hard problem. I first coded it in a very naïve way, just to see if everything worked. Fortunately, the number of graphs that we should find are documented on the internet. It is OEIS series A000109 : "1, 1, 1, 2, 5, 14, 50, 233, 1249, 7595, 49566, 339722, 2406841, 17490241, 129664753, 977526957, 7475907149, 57896349553, 453382272049, 3585853662949".

I first missed rule 3. This means that for 12 nodes I found 49565 different graphs. In other words, I missed one graph : the icosahedron. If instead I would have read the article "generation of triangulations of the sphere" from 1965 ! then I would have found it at page 3.

I took my program about a day to generate 49566 graphs with 12 nodes. Plantri does the same in less than a second. In fact, they have made their effort available on the internet : https://cs.anu.edu.au/people/Brendan.McKay/plantri .

I tried to edit plantri and adapt it to my needs. However this program is optimized to prevent a rule transition if it is known that it will not produce a new MPG. So while I succeeded in "recording" the transitions that plantri did, it turned out that I missed the bulk of them (which of course is the whole idea behind that program).

In a way this was a relief because it meant that I had found *why* my program was so slow. It was not *inherently* slow. I just did many more calculations because I *needed* these extra calculations.

Nevertheless I rewrote my program. There are plenty of ways to represent a graph. However, I chose to represent graphs as faces. Each Face has a list of its neighbor faces. The program is written in c# and might be made about 2 orders of magnitude faster by coding it in c++. I have not done that because c+ is 20 years behind me now. But I am still considering it.

With this representation, rule1 becomes this :

```
foreach (var f in Faces)
 {
     var clone = Clone();


     var clonedFace = clone.GetFaceById(f.FaceId);

     var newVertex = clone.NextFreeVertexId();

     var id = clone.NextFreeFaceId();


     // create 3 new faces and replace old with new in neighbours of graph

     var ne = CreateRule1SubFace(clonedFace, clonedFace.Vertex1, clonedFace.Vertex2,
newVertex, ref id);

     var nw = CreateRule1SubFace(clonedFace, clonedFace.Vertex1, clonedFace.Vertex3,
newVertex, ref id);

     var south = CreateRule1SubFace(clonedFace, clonedFace.Vertex2, clonedFace.Vertex3,
newVertex, ref id);


     // add 3 new faces

     clone.AddFace(ne);

     clone.AddFace(nw);

     clone.AddFace(south);


     // make them neighbours of each other

     ne.AddNeighbour(nw);

     ne.AddNeighbour(south);


     nw.AddNeighbour(ne);

     nw.AddNeighbour(south);


     south.AddNeighbour(ne);

     south.AddNeighbour(nw);


     // remove old face

     clone.RemoveFace(clonedFace);

     yield return clone;

 }

private Face CreateRule1SubFace(Face oldFace,
   int oldVertex1, int oldVertex2, int newVertex, ref int id)
```

```
{
 var newFace = new Face(oldVertex1, oldVertex2, newVertex) {FaceId = id};
 id++;
 foreach (var neighbour in oldFace.Neighbours)
 {
     if (neighbour.ContainsEdge(oldVertex1, oldVertex2))
     {
         neighbour.ReplaceNeighbour(oldFace, newFace);
         break;
     }
 }
 return newFace;
}
```

Rule 2 is also quite readable :

```
var alreadyDone = new SortedSet<Edge>();

 foreach (var leftFace in Faces)
 {
     foreach (var rightFace in leftFace.Neighbours)
     {
         var common = leftFace.CommonEdge(rightFace);

         // avoid duplicate work this can happen since we attack each Face which means
         // we visit each edge twice.
         if (alreadyDone.Contains(common)) continue;
         alreadyDone.Add(common);

         int top = common.Vertex1, bottom = common.Vertex2;

         // start new instance
         var clone = Clone();
         var nextFaceId = clone.NextFreeFaceId();

         var newVertex = clone.NextFreeVertexId();

         var leftFaceCloned = clone.GetFaceById(leftFace.FaceId);

         var rightFaceCloned = clone.GetFaceById(rightFace.FaceId);


         leftFaceCloned.RemoveNeighbour(rightFaceCloned);

         rightFaceCloned.RemoveNeighbour(leftFaceCloned);


         var leftVertex = leftFaceCloned.RemainingVertex(common);

         var rightVertex = rightFaceCloned.RemainingVertex(common);


         // create the faces

         var leftTop = new Face(newVertex, top, leftVertex){FaceId = nextFaceId++};

         var leftBottom = new Face(newVertex, bottom, leftVertex) ){FaceId =
nextFaceId++};

         var rightTop = new Face(newVertex, top, rightVertex) ){FaceId = nextFaceId++};

         var rightBottom = new Face(newVertex, bottom, rightVertex) ){FaceId =
nextFaceId++};


         // add faces

         clone.AddFace(leftTop);

         clone.AddFace(leftBottom);

         clone.AddFace(rightTop);

         clone.AddFace(rightBottom);


         // fix the neighbours among the new

         leftTop.AddNeighbour(leftBottom);

         leftTop.AddNeighbour(rightTop);


         leftBottom.AddNeighbour(leftTop);

         leftBottom.AddNeighbour(rightBottom);



         rightTop.AddNeighbour(rightBottom);
         rightTop.AddNeighbour(leftTop);

         rightBottom.AddNeighbour(rightTop);
         rightBottom.AddNeighbour(leftBottom);

         // fix the old neighbours left
         foreach (var neigh in leftFaceCloned.Neighbours)
         {
             if (neigh.ContainsEdge(top, leftVertex))
             {
                 neigh.ReplaceNeighbour(leftFaceCloned, leftTop);
                 continue;
             }
             if (neigh.ContainsEdge(bottom, leftVertex))
             {
                 neigh.ReplaceNeighbour(leftFaceCloned, leftBottom);
                 continue;
             }

             throw new AccessViolationException();
         }

         // fix the old neighbours right
         foreach (var neigh in rightFaceCloned.Neighbours)
         {
             if (neigh.ContainsEdge(top, rightVertex))
             {
                 neigh.ReplaceNeighbour(rightFaceCloned, rightTop);
                 continue;
             }
             if (neigh.ContainsEdge(bottom, rightVertex))
             {
                 neigh.ReplaceNeighbour(rightFaceCloned, rightBottom);
```

```
                    continue;
                }
                throw new AccessViolationException();
            }



            clone.RemoveFace(rightFaceCloned);
            clone.RemoveFace(leftFaceCloned);

            yield return clone;
        }
    }
```

And finally here is rule 3 :

```
foreach (var leftFace in Faces)
 {
     foreach (var middleFace in leftFace.Neighbours)
  {
      var commonLeft = leftFace.CommonEdge(middleFace);

      foreach (var rightFace in middleFace.Neighbours)
      {
          if (rightFace.FaceId == leftFace.FaceId) continue;
          var commonRight = middleFace.CommonEdge(rightFace);

          var topVertex = commonRight.SharedVertexWith(commonLeft);
          if (topVertex<0) continue;

          var vertex1 = topVertex;
          var vertex3 = commonRight.OtherVertex(vertex1);
          var vertex4 = commonLeft.OtherVertex(vertex1);
          var vertex2 = rightFace.RemainingVertex(new Edge(vertex1, vertex3));
var vertex5 = leftFace.RemainingVertex(new Edge(vertex1, vertex4));

var clone = Clone();

var nextFaceId = clone.NextFreeFaceId();
var newVertex = clone.NextFreeVertexId();


var leftFaceCloned = clone.GetFaceById(leftFace.FaceId);
var middleFaceCloned = clone.GetFaceById(middleFace.FaceId);
var rightFaceCloned = clone.GetFaceById(rightFace.FaceId);

leftFaceCloned.RemoveNeighbour(middleFaceCloned);
rightFaceCloned.RemoveNeighbour(middleFaceCloned);
middleFaceCloned.RemoveNeighbour(leftFaceCloned);
middleFaceCloned.RemoveNeighbour(rightFaceCloned);


var face1 = new Face(newVertex, vertex1, vertex2) { FaceId = nextFaceId++ };
var face2 = new Face(newVertex, vertex2, vertex3) { FaceId = nextFaceId++ };
var face3 = new Face(newVertex, vertex3, vertex4) { FaceId = nextFaceId++ };
var face4 = new Face(newVertex, vertex4, vertex5) { FaceId = nextFaceId++ };
var face5 = new Face(newVertex, vertex5, vertex1) { FaceId = nextFaceId };

// add faces
clone.AddFace(face1);
clone.AddFace(face2);
clone.AddFace(face3);
clone.AddFace(face4);
clone.AddFace(face5);

// add the neighborhood among the new faces
face1.AddNeighbour(face2);
```

```
face2.AddNeighbour(face3);
face3.AddNeighbour(face4);
face4.AddNeighbour(face5);
face5.AddNeighbour(face1);

foreach (var neigh in leftFaceCloned.Neighbours)
{
    if (neigh.ContainsEdge(vertex1, vertex5))
    {
        neigh.ReplaceNeighbour(leftFaceCloned, face5);
        continue;
    }
    if (neigh.ContainsEdge(vertex4, vertex5))
    {
        neigh.ReplaceNeighbour(leftFaceCloned, face4);
        continue;
    }
    throw new InvalidOperationException();
}

foreach (var neigh in middleFaceCloned.Neighbours)
{
    if (neigh.ContainsEdge(vertex3, vertex4))
    {
        neigh.ReplaceNeighbour(middleFaceCloned, face3);
        continue;
    }

    throw new InvalidOperationException();
}

foreach (var neigh in rightFaceCloned.Neighbours)
{
```

```
                if (neigh.ContainsEdge(vertex2, vertex3))
                {
                    neigh.ReplaceNeighbour(rightFaceCloned, face2);
                    continue;
                }
                if (neigh.ContainsEdge(vertex1, vertex2))
                {
                    neigh.ReplaceNeighbour(rightFaceCloned, face1);
                    continue;
                }
                throw new InvalidOperationException();
            }

            clone.RemoveFace(rightFaceCloned);
            clone.RemoveFace(middleFaceCloned);
            clone.RemoveFace(leftFaceCloned);
            clone.CheckSanity();
            clone.Title = Title + " <3>" + commonLeft + "*" + commonRight;

            yield return clone;
        }
```

10 References :

- Fast generation of planar graphs by Gunnar Brinkmann and Brendan D. McKay
- The boost Graph library
- The chromatic polynomials and chromaticiy of graphs . Gong, Koh & Teo.
- The fouor color theorem, Saaty and Kainen.
- Computing chromatic polynomials for special families of graphs, Loering Beatrice
- Handbook of graph drawing an visualization, Tamassia
- Graph theory and its applications, Jonathan L. Gross, Jay Yellen
- Graphs, Colourings and the four-colout theorem, Robert a. Wilson
- Graph algorithms, Shimon Even and guy Even.
- Counting on frameworks, Jack E. Graver.

- Algorithmic graph theory and perfect graphs, Martin Charles Columbic.
- The knot book, Colin C. Adams
- The theory of graphs, Claude berge
- Graph Theory As I Have Known It, Tutte
- Efficient Algorithms for Listing Combinatorial Structures, Leslie Golberg
- Pearls in Graph Theory: A Comprehensive Introduction, Nora Hartsfield
- Modern Graph Theory, Bollobas, Bela
- Algebraic Graph Theory, Godsil, Chris
- Discrete And Combinatorial Mathematics: An Applied Introduction, R. Grimaldi
- Planar Graph Isomorphism is in Log-Space, Samir, Datta Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, Fabian Wagner.
- Subgraph Isomorphism in Planar Graphs and Related Problems, David Eppstein
- Is the four-color conjecture almost false, Sami Beraha, Joseph Kahane.
- How to draw a graph, Tutte
- A simple linear time algorithm for embedding maximal planar graphs, Hermann Stamm-Wilbrandt.
- Generation of triangulations of the sphere, Robert Bowen and Stepehn Fisk.