LEFT RIGHT PLANARITY TESTING AND MAXIMAL PLANAR SUBGRAPH

by

Ibrahim Gurgil

A Dissertation Submitted in Partial Fulfillment

of the Requirements for the Degree of

Doctor of Philosophy in Computational Sciences

Middle Tennessee State University

December 2019

Dissertation Committee:

Prof. Chris Stephens, Chair

Prof. Dong Ye

Prof. Xiaoya Zha

Prof. Suk Jai Seo

Acknowledgements

I would like to thank Dr. John Wallin for his support and guidence. I would like to give my thanks to Dr. David Christopher Stephens for his patience and directions. I would lik to thank my committee members Dr. Dong Ye, Dr. Xiaoya Zha, and Dr. Suk Jai Seo. I would like to thank my long-term advisor Dr. Ismail Gok from Ankara University.

I would like to thank the Turkish Ministry of National Education for giving me this oppurtunity.

I would like to thank my friends, especially Yusuf Helvali and Harun Resit Kaya for being there for me whenever I need.

I would like to thank my best friend, my wife and mother of my child, Suzan Odabasi Gurgil, for being everthing for me. I would like to thank Arya Ipek Gurgil for being my sunshine.

Abstract

According to our best knowledge, the Left-Right(LR) algorithm offers the fastest speed for planarity testing. We show that in finding the Kuratowski (K_5) graph, the LR algorithm fails in some cases. We have revised the LR algorithm for the Kuratowski graph with 5 vertices. We further added some features to the LR algorithm to extract the Kuratowski subgraphs. We will discuss the additions to the LR approach and its possible usages.

It is proven that the LR approach is one of the fastest algorithms to find if a graph is planar. We updated the algorithm of LR to extract the Kuratowski subgraphs efficiently. A maximal planar subgraph G' of G is a graph with the same vertex set of G and with a minimal set of missing edges from G such that if any removed edge is added to G', the graph becomes non-planar again. If G is planar, then the maximum (maximal) planar subgraph is itself. Otherwise we will discuss how to find the minimum set of edges to be removed to make a graph planar using the updated LR approach.

TABLE OF CONTENTS

List of Figures

Chapter 1

INTRODUCTION

Nice drawings of sparse non-planar graphs can be achieved by determining a maximum planar subgraph (MPS) and then an augmenting of this graph [19]. The MPS question has been studied extensively in computer science research, and has some applications [21, 25] in circuit layouts. Facility layout problems also have strong connections with MPS question. Turan has proposed the minimum crossing question for complete bipartite graphs, which can be achieved with MPS.

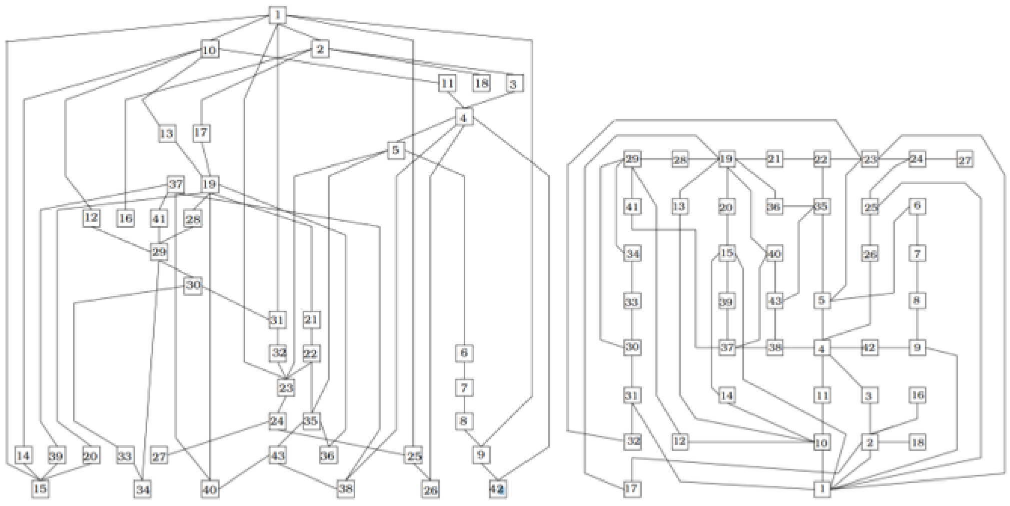


Figure 1: Two different drawings of same implementation

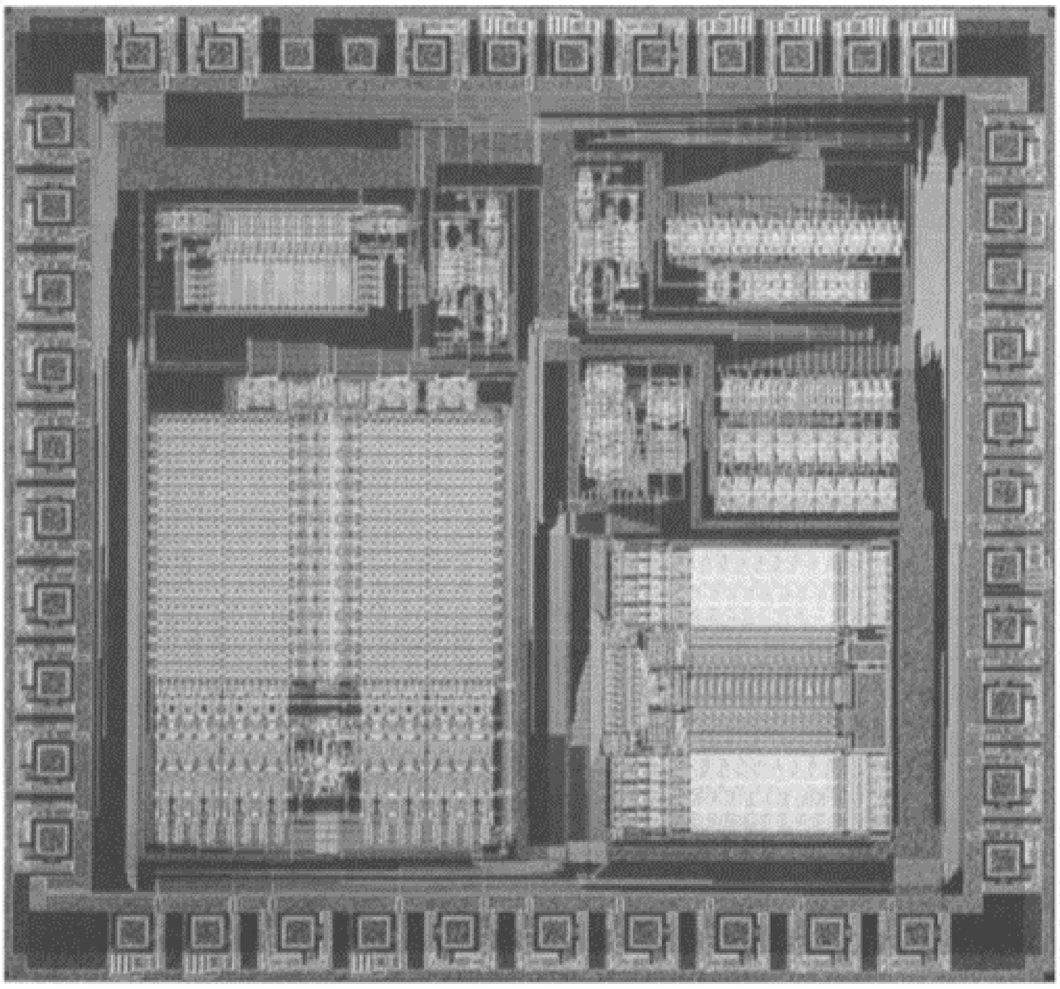


Figure 2: A circuit layout needs to have minumum crossing, which can be achieved by MPS question. [28]

The first implementation (on the left) is introduced in a paper about automatic graph drawing by Tamassia, Di Battista and Batini [27] and the number of crossings has been decreased to 9 (by removing 4 edges out of 62, then adding them back) by M.Junger and P.Mutzel[19] using the branch and cut technique.

The Left Right (LR) approach was introduced by [9]. They originally used an idea from [26]. The LR approach was improved in [10] and [11]. H. de Fraysseix and P. Ossona de Mendez have published a series of papers in [8, 6, 12] to simplify the approach, and they used the LR approach to isolate the Kuratowski subdivision if the graph is not planar. Otherwise there seem to have been no significant improvements in the LR approach.

Most of the developments in the field of planarity testing have been made in the vertex-addition, edge-addition, and path-addition approaches.

The vertex-addition approach was proved in the mid 1960s to have polynomial timing [20]. In 1976 the approach was improved to linear timing in the number of edges[15] .

The edge-addition approach was inspired by the PQ-tree implementation of the vertex-addition method[1] .

The path-addition approach was first published by [18]. In the mid 1990s, LEDA was introduced based on path-addition; see [24, 23, 22].

In 2012, the original authors of the LR algorithm published a new approach based on the LR approach [7]. The 2012 paper simplified the planarity-testing algorithm, and simplified the extracting of the Kuratowski subgraph. The LR algorithm is, according to our best knowledge, the simplest algorithm. It is the fastest algorithm to test for planarity [2] and can be used to extract the Kuratowski subgraph.

One reason the LR algorithm is not popular is the absence of an easy proof of the algorithm. Most other algorithms depend on Kuratowski subgraphs; therefore their correctness is tied to the well-known characterization of planarity from textbooks, and their implementations are easier to understand. To address this issue, the original LR authors corrected some minor issues and even created a third approach to attack planarity testing [12]. Their second approach is also derived from the LR approach and can be found at PIGALE [5].

For the maximal planar subgraph problem, we know [16] that finding the minimum set of edges to be removed from a graph to get a planar subgraph is an NP-hard problem. In this project we attack the problem with respect to edge removal; however, there are also some investigations along the lines of vertex removal [13]. A similar approach to ours has been used by [14], by adding edges one by one to the Depth-First Search (DFS) tree. His approach finds the maximal planar subgraph with approximation in linear time and also finds only one maximal subgraph at a time. There is also a comparison between different approaches (by edge removal) in [3]. Chimani, Mutzel, and Schmidt [4] used the method of Boyer and Myrvold [1] to extract multiple Kuratowski. By the conclusion of [17], Kuratowski subdivision methods are better than methods of simulating facial walks and total orders.

First, in contrast to the edge-addition, vertex-addition, and path-addition methods, which are mainly focused on Kuratowski extraction, the LR method focuses on planarity alone. While LR planarity testing is found to be the fastest among these, its usage is limited. The LR method has inherently a missing piece, such that by adding it we can make the LR method more useful. Planarity testing is important for planar drawing. One can understand by planarity testing whether a graph can be drawn in the plane. Our first main goal for this project is to correct the algorithm, and thereby allow the LR approach to become more widespread.

Our first question is why the LR approach is not used by researchers. What is the main problem behind this absence? While it is just a planarity testing algorithm, where can we use the LR approach in a meaningful way?

Our second question is to find all maximal planar subgraphs. Moreover, we wish to find all maximal planar subgraphs by using a mixture technique mainly based on the LR approach with the fixed DFS tree.The LR approach is the fastest known amongst the planarity testing approaches. By doing this we hope to benefit the Kuratowski extraction methods.

For a given undirected and simple graph G=(V,T $\cup$ B), where T is the set of edges appear at depth-first-search step and other edges in set B, the LR approach asks the question if the set of edges could be divided into two group, left and right. The idea behind the LR algorithm is , if every edge of B , can connect its endpoints. Since the algorithm looks to the edges one by one, when we find the contradiction (crossing), we stop the algorithm.While other algorithms result in a Kuratowski subgraph and their result in planarity builted in this way, their computational load is more than the LR algorithm.

Planarity testing in linear time has been known for some time and has been used to solve many contemporary issues. While the LR approach is also a planarity testing algorithm, since its main goal is planarity (whereas other approaches have mostly used the Kuratowski theorem), other than the original LR authors, not many people have focused on it (even though the proof is easier than those of the other approaches).

With our updates, we hope to fill that gap. This approach is the fastest amongst known planarity tests. All approaches have linear time, but the LR produces the fastest results.

In terms of storage space, the LR approach is interested in a whole graph at a time. Therefore it needs more space than other approaches because other approaches focus on the part of the graph that they just added. For some bigger graphs or some special graphs (for example, very dense or very sparse graphs), all the algorithms still have some problems.

Second, the maximal planar subgraph problem is a question which has been studied for a long time. There are many methods using heuristic approaches, and lately researchers have focused on the Integer Linear Programs (ILP) which give the exact solutions at the cost of time. ILP approaches mostly used branch-cut techniques. To the best of my knowledge, they get only one maximum planar subgraph at first; then to get other maximum planar subgraphs, the input (or the procedure that controls the input) must be manipulated and the program must be re-run. If the graph is not very well-known (for instance, if the graph is anything other than complete or complete bipartite), getting all the maximum planar subgraphs most of the time is not even possible.

1.1 Planarity Testings

There are two popular definition of planar graph, and we have also focused on these two:

1. A graph is planar if it can be drawn in a plane without crossings of its edges except the endpoints.

2. A graph is planar, if and only if there is no Kuratowski subgraph of that graph.

Theorem 1: (Kuratowski's theorem) A graph is planar if and only if there is no subgraph is a division of K_5 or $K_{3,3}$

Theorem 2: (Wagner's theorem) A graph is planar if and only if it does not contain a minor that is isomorphic to K_5 or $K_{3,3}$

Euler's Formula: If a graph G=(V,E) and has a f faces, then n-m+f=2, where n and m are number of vertices and edges , respectively.

If a graph is planar, with the n vertices and m edges, then $m \leq 3n - 6$. It is a well-known result of Euler's formula and has been used in the planarity testings at the very beginning, if there is any need to test it or not.

For a given simple and undirected graph G=(V,E), deciding the planarity of a graph has been researched for the last decades. While there are other definition of planar graphs, mostly these two has been used.

1.2 Left Right Planarity testing

While the proof of the all planarity testings are very hard and includes many implementations before hand, the LR has a very basic idea behind its proof.

For a given graph G with it's corresponding Depth-First-Search tree, if every vertex in the DFS tree could be closed by left or right. If a vertex v is closed from left and right from vertices v_i to v_j, then there can not be any vertex connects the vertex v to any other vertex higher than the highest of v_i and v_j(or lower than the lowest of v_i and v_j) . As we see in the figure 3, since vertex number 3 has been closed from left and right in both (3a) and (3b) by vertex 5 (also vertex 4 but as we see 5 is higher than 4 in the DFS tree), there can not be an edge between vertex 6 and vertex 3, which yields to conradiction if we have an edge from vertex 6 to vertex 3. Please note that (3a) and (3b) in the figure 3 are both $K_{3,3}$ without the (6,3).

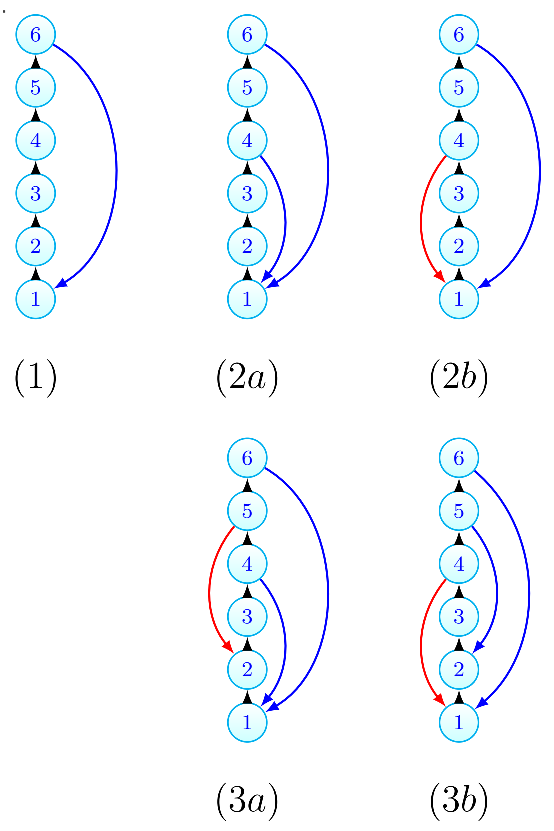
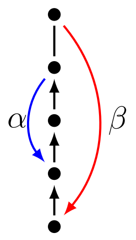
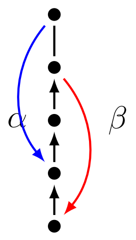


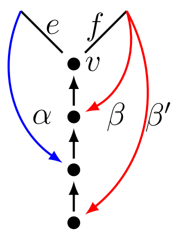
Figure 3: Figures for adding edges to $K_{3,3}$



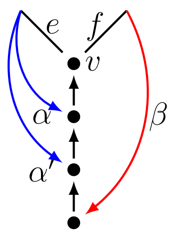
Type(i): $[\alpha, \beta]\epsilon$S
whenever there exists $\beta\epsilon$B such that
$v^-(\beta) < v^-(\alpha) < v^+(\alpha) < v^+(\beta)$



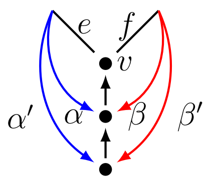
Type(ii): $[\alpha, \beta]\epsilon$D
$v^-(\alpha) < v^-(\beta) < v^+(\alpha) < v^+(\beta)$



Type(iii): $[\alpha, \beta]\epsilon$D
if there exist a e,fϵ T and
$\beta\epsilon$ B such that
$\alpha\epsilon$ B(e) and $\beta, \beta'\epsilon$ B(f) with
$v^-(\beta') < v^-(\alpha) < v^-(\beta) < v^-(e) = v^-(f)$



Type(iv): $[\alpha, \beta]\epsilon$S
if there exist a e,fϵ T and
$\beta\epsilon$ B such that
$\alpha, \alpha'\epsilon$ B(e) and $\beta\epsilon$ B(f) with
$v^-(\beta) < v^-(\alpha') \leq v^-(\alpha) < v^-(e) = v^-(f)$



Type(v): $[\alpha, \beta]\epsilon$D
if there exist a e,fϵ T and $\beta', \alpha'\epsilon$ B such that
$\alpha, \alpha'\epsilon$ B(e) and $\beta\beta'\epsilon$ B(f) with
$v^-(\alpha') = v^-(\beta') < v^-(\alpha) = v^-(\beta) < v^-(e) = v^-(f)$

Figure 4: α and β in different and same partitions .

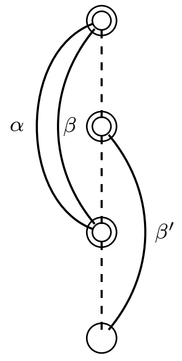
These five types of graphs have been introduced by the original authors in the paper of 1982. They are investigated later, and resulted in similar results with having the three graphs in figure 4.

Later on these 3 graphs have been put into one algorithmic formula.

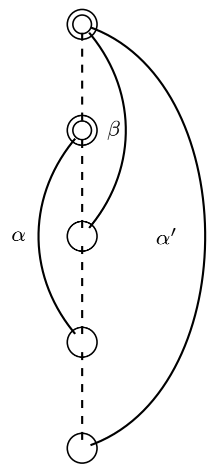
Afterwards, Left Right planarity testing is usually formulized as:

- All return edges of e_1 strictly higher than lwp(e_2) belong to one partition.
- All return edges of e_2 strictly higher than lwp(e_1) belong to other partition.

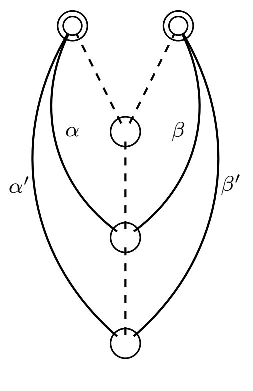
If the edge set can be divided into the two partitions, then it is planar; otherwise it is non-planar.



Type(i): $[\alpha, \beta]\epsilon$S
if there exist a $\beta'\epsilon$ B and
$\alpha, \beta\epsilon$ B such that
$v^-(\beta') < v^-(\alpha), v^-(\beta) < v^+(\beta') < v^+(\alpha), v^+(\beta)$



Type(ii): $[\alpha, \beta]\epsilon$D
if there exist a $\alpha'\epsilon$ B and
$\alpha, \beta\epsilon$ B such that
$v^-(\alpha') < v^-(\alpha) < v^-(\beta) < v^+(\alpha) < v^+(\beta), v^+(\beta')$



Type(iii): $[\alpha, \beta]\epsilon$D
if there exist a $\alpha', \beta'\epsilon$ B and
$\alpha, \beta\epsilon$ B such that
$v^-(\alpha') = v^-(\beta') < v^-(\alpha) = v^-(\beta) < v^+(\alpha'), v^+(\beta')$

Figure 5: α and β in different and same partitions

Chapter 2

LEFT RIGHT PLANARITY TESTING

In this section, we will look into the LR testing in details. For this, we will introduce some definitions.

2.1 Original Left Right planarity testing

As we mentioned earlier, LR testing is invented by Fraysseix and Mendez at 1982. And they defined the testing with respect to angles of the edges at the Depth-First-Search tree. After that, authors changed the definition based on angles to a definition based on the edges.

2.1.1 Depth First Search algorithm

DFS on a graph G gives us a tree T, which starts from a random vertex of G and visits edges of G, searching for new edges to visit from the last vertex visited to an unvisited vertex, until we have visited all vertices of G or run out of edges to unvisited vertices. When we get the end we do backtracking until there is no vertex left unvisited.

.

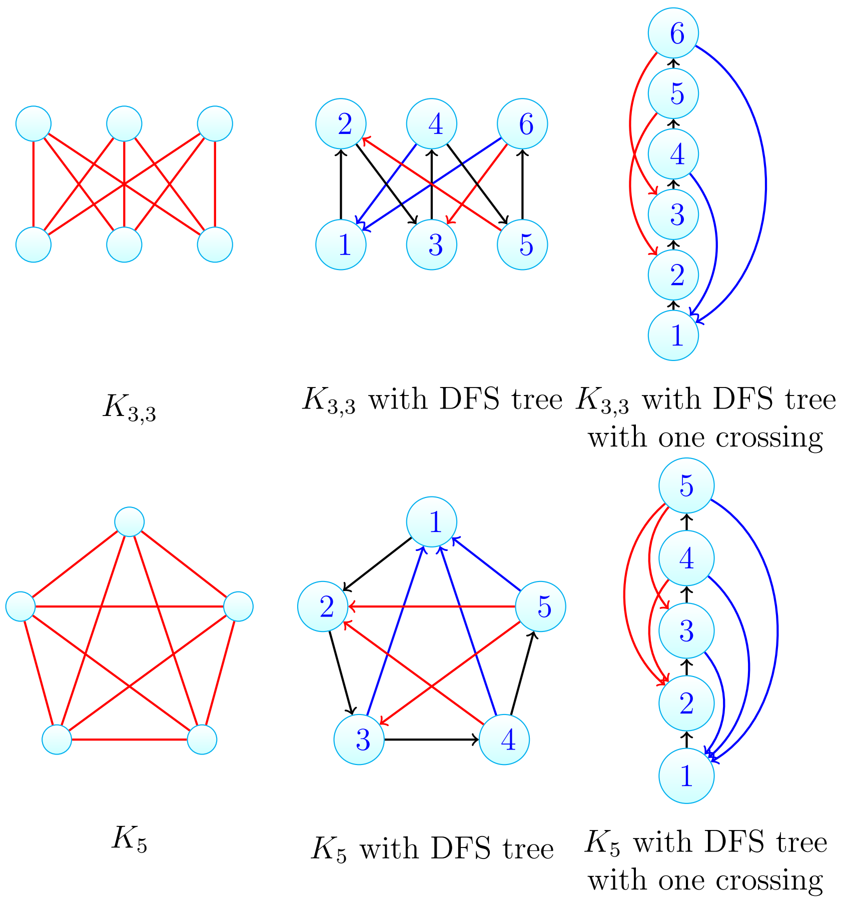

$K_{3,3}$
$K_{3,3}$ with DFS tree
$K_{3,3}$ with DFS tree with one crossing
K_5
K_5 with DFS tree
K_5 with DFS tree with one crossing

Figure 6: Kuratowski graphs with DFS, and DFS tree.

2.1.2 Tremaux tree

Tremaux tree, also called as DFS tree, is a tree with a directed edges discovered in DFS procedure and addition of rest of the edges.

For a graph G and with its responding DFS procedure. We call any visited edge, during the DFS procedure, e=(u,v) a tree edge and orient e from the first visited vertex of e to the second. The remaining edges are called cotree edges or back edges, and we orient them from the later visited vertex to the earlier visited vertex. For detailed information, see [9] or [7].

2.1.3 Definitions of LR testing

We always consider a finite, simple, connected, undirected and unweighted graph G=(E,V) with edge set E and vertex set V. The number of edges will be $|E| = m$, and number of vertices $|V| = n$.

Definition 1: The height of a vertex with respect to the DFS tree is its distance (in terms of the number of edges) to the first vertex of the DFS tree. We show the height of vertex v as height(v).

Definition 2: (Low order) For u,vϵV, if height(u) is less than height(v), and they share the DFS tree path of vertex u, then we say u is lower than v, written as u<v. We also say vertex v is higher than vertex u.

Definition 3: The low point of an edge is defined differently for tree edges and cotree edges, as follows.

• The low point of a cotree edge e=(u,v) (where e is directed from u to v) is v.

• The low point of a tree edge e=(u,v) (where e is directed from u to v) is defined to be x such that for a vertex $x \leq y$, there is a cotree edge e'=(y,x) and $x < u$. In other words, there is a cycle which starts from x, then goes through e=(u,v) and ends at x again. We write it as low(e)= x.

Definition 4: The return edge of a cotree edge is itself, and the return edge of a tree edge e=(u,v) to be e'=(y,x) where x<u, and v$\leq$y

Definition 5: (Fork) An outgoing cotree edge e* from a vertex v is called a back edge (or cotree edge). All back edges of a vertex v will be considered as one set, while all tree edges going out from vertex v are considered as a set individually. If there is more than one such set from a vertex v, we call that vertex a fork.

Definition 6: The lowest point of a set E is the lowest of the low points of its members. We will denote this as lwp(E).

.

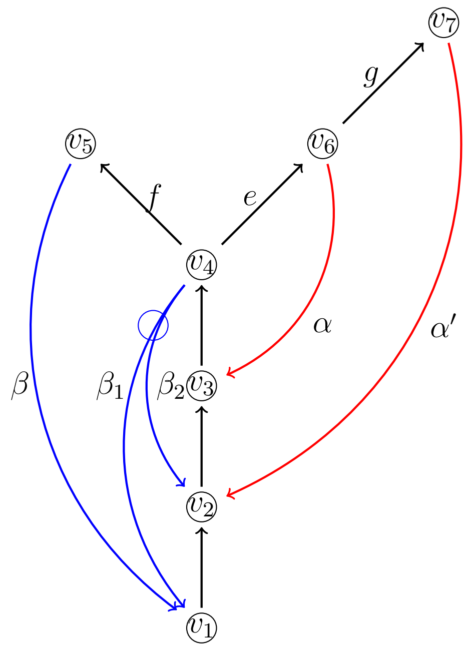


Figure 7: A graph for definitions

In the figure 7,

Tree edges are : e, f and g;

Back (cotree) edges are : $\beta, \beta_1, \beta_2, \alpha, and\alpha'$;

Low order : $v_4 < v_5$ and $v_4 < v_6$ but v_5 and v_6 can not be comparable;

low(e)=$\{v_2, v_3\}$,low(f)=v_1 and low(α)=v_3;

lwp(e)=v_2

return(e)=$\{\alpha, \alpha'\}$

Outgoing edges of the vertex v_4 are e, f, and the set $\{\beta_1\beta_2\}$. Since there is more than one outgoing edge, we call v_4 a fork. v_6 is also a fork since it has g and α as outgoing edges.

2.1.4 Linear time testing and algorithm

After these preliminaries, we are ready to give the definition. For a given fork v, assume e and f are the two outgoing edges, then:

All the return edges of e, which are higher than the lowest point of f belongs to one partition;

All the return edges of f, which are higher that the lowest point of e belongs to the other partition.

If we can divide all back edges (that satisfy these two conditions) to two partitions after processing all the forks, than the graph G is planar. Otherwise, G is non-planar.

DFS procedure and linear time algorithm follows as:

```
% Start with a random vertex , root's main edge is empty.
DFS($v$)
    $e = main\_edge(v)$ % for the lowpoints of the tree edges.
    $while$ there is an not-oriented, outgoing edge $\{v, w\}$ of $E$
        $orient$ $\{v, w\}$ as $(v, w)$
        $lwpt((v, w)) = v$ % assign the lowpoint.
        $if$ $height(w) == \infty$ % tree edge
            $main\_edge(w) = (v, w)$ % assign the main edge.
            $height(w) = height(v) + 1$ % calculate the height
            $DFS(w)$ % first go to a leaf,than back
        $else$ % back edge
            $lwpt((v, w)) = w$ % change the lowpoint
        end
        $if$ $lwpt((v, w)) < lwpt(e)$
            $lwpt(e) = lwpt((v, w))$ % update the lowpoint
```

Figure 8: DFS procedure

```
% if there is more than one outgoing edge from $v$,$E^+(v)$
$for$ $\forall$ pair $(e_i, e_j)$ $\epsilon$ $E^+(v)$
    $for$ $\forall$ $b$ $\epsilon$ $E^+(e_i)$ % b is a back edge from $e_i$
        $if$ $lwpt(b) < v$ and $lwpt(b) > lwpt(e_j)$
            $if$ $flag(e_j) == 0$
                $if$ $sign(b)$==-1; $then$ $flag(e_i) = -1; flag(e_j) = 1;$
                $else$ ($if$ sign(b)==1 | 0) $flag(e_i) = 1; flag(e_j) = -1;$
            $else$
                $if$ $sign(b) == 0$ then; $sign(b) == flag(e_i);$
                $else$ $if$ $sign(b) * flag(e_i) == -1$ then;
                    $contradiction$
```

Figure 9: LR testing

2.2 A counter example for Left Right testing

While working on some other questions, we needed a simple and fast planarity testing to implement our problem. We started to work on the LR algorithm, and we got incorrect results. When we looked deeper into the algorithm and papers again, we discovered that almost all graphs that had been used to check the algorithm are similar. While the algorithm works for these type of graphs, it has inheredently a missing part. Here is the example:

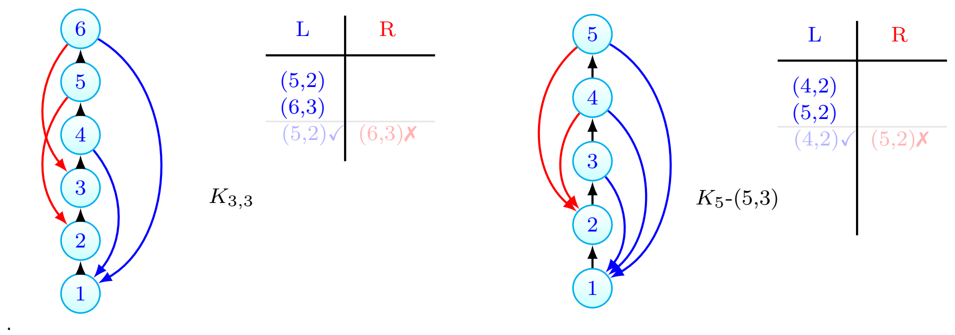


Figure 10: Comparing the $K_{3,3}$ with K_5 minus an edge. As we expected $K_{3,3}$ has a contradiction; but K_5-(5,3) also has an unexpected contradiction.

We know that both graphs are Kuratowski graphs which are non-planar. For $K_{3,3}$, we get the desired result with these original two conditions. For the K_5, we should not hit non-planarity until the end, but if one looks at the LR table, (4,2) and (5,2) have been put in the left side, and for the next fork, (4,2) has been put in the left side. By these conditions we should put the (5,2) on the right side, but since (5,2) is already in the left side it is a contradiction. This is similar to the situation for $K_{3,3}$, in which (5,2) and (6,3) have to be in the same partition for one fork, and different partition for the other fork so there is a contradiction. Thus for $K_{3,3}$,the LR testing stops and produce a non-planar result.

2.3 Filling the missing part of Left Right testing

When we look into this, we see that while equality has been managed by a graph in the theory (see Figure 5, Type[iii]) if there is more than one outgoing tree edge from our fork, if there is one tree edge and a back edge(s) from the fork, equality has not been considered.

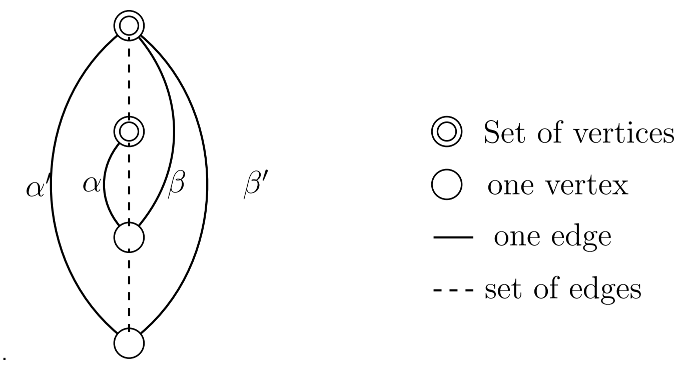


Figure 11: Missing part of LR testing

To overcome this counter example, we add an extra condition and with that condition, we have corrected the algorithm.

2.3.1 A new procedure for Left Right algorithm

To overcome this issue, I have added the following step:

• If the one of the sets is formed by cotree edges (say E_1) , and if the lwp(e_1)=lwp(e_2) and low(e_i)=low(e_j) where e_i is from E_1, and e_j is from E_2 and also low(e_i) $> lwp(e_2)$ and low(e_j) $> lwp(e_1)$, they do not cause a contradiction.

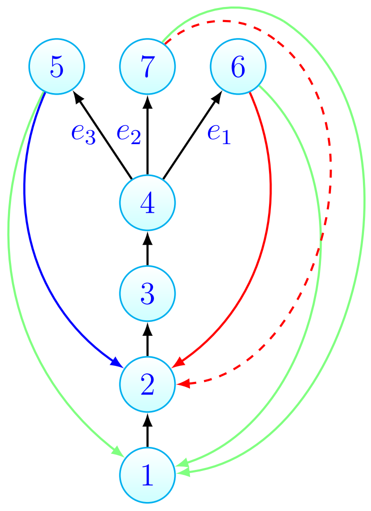
• If there is a second edge at the partition of E_2, say e_k , and the first step is also there if low(e_k) $> low(e_1)$ then it is a contradiction, even if e_k did not appear at the partition of e_1.

```
% if there is no back edges from fork, do original!!
% if there is more than one outgoing edge $(e_i, e_j)\epsilon\ E^+(v)$ and $e_i \epsilon B(v)$
$for\ \forall$ pair $(e_i, e_j)\ \epsilon\ E^+(v)$
    $for\ \forall\ b\ \epsilon\ E^+(e_i)$ % b is a back edge from $e_i$
        $if\ lwpt(b) < v$ && $lwpt(b) > lwpt(e_j)$
            $if\ flag(e_j) == 0$
                $if\ sign(b)$==-1; $then\ flag(e_i) = -1; flag(e_j) = 1;$
                $else\ (if$ sign(b)==1 | 0) $flag(e_i) = 1; flag(e_j) = -1;$
            $else$
                $if\ sign(b) == 0$ then; $sign(b) == flag(e_i);$

                $if\ sign(b) * flag(e_i) == -1$ then;
                    $if$ lwpt(b) $> lwpt(b_m)$ and $high(b) > v$
                        contradiction
                    $else$ crit($e_i$)=1;
                $else\ if$ crit($e_i$)==1 then
                    $if$ lwpt(b) $> lwpt(b_m)$ and $high(b) > v$
                        contradiction
```
.

Figure 12: New LR testing

Note that this issue appears only if one side of the fork is coming from back edges (or for each fork one). Consider the next figure;



	L	R
(e_1, e_3)	(5,2)	(6,2)
(e_2, e_3)	(7,2)	(6,2)✓
(e_1, e_2)	(5,2)✓	(7,2)✗

.

Figure 13: Same problem which causes contradiction.

Consider the sets of $(5, 6, 7)$ and $(1, 2, 4)$ vertices , so the graph has a $K_{3,3}$ subgraph, so it is non-planar.Because, none of the (e_1, e_2, e_3) is a back edge. Also note that, to have a contradiction, one fork is enough.

2.3.2 The computational complexity

Since the LR testing algorithm has been introduced as linear timing respect to edges, although there is a new procedure in the test, the procedure is still based on the number of edges if the "fork" has back edge(s). If "fork" does not have a back edge, then original program holds. So timing is still linear respect to number of edges.

Chapter 3

MAXIMAL PLANAR SUBGRAPH QUESTION

Since Maximal Planar Subgraph(MPS) has been studied for a while, and although there has been some algorithms presented but not any of them used the LR approach. For MPS question, there are two mainstreams are focused. First by edge adding/removing edges with heuristic methods, since this way has faster, it does not guarante the finding optimal solution. Second method is focused in Kuratowski subgraphs, called ILP, and they are using the planarity testing based on Kuratowski subgraphs, then by removing Kuratowski subgraphs, they aim to find MPS. Their success is for sure, but they only focus on one (or a few) MPS and their timing is worse than first mainstream mentioned.

Our goal is here is to combine the two mainstreams with using LR approach. Our heuristic method is based in the Kuratowski subgraph patterns that showed up in the LR partitioning. By using the idea behind the ILP methods, we aim to get MPS and also possible solutions for maximum planar subgraph question.

3.1 Patterns

Since the LR approach have not been studied very deeply, we have realized that there are some patterns repeats during the Kuratowski subgraph trials. Then we expand our research to this area and see that we have 4 basic patterns (and their reflections) that responds to Kuratowski subgraphs. With the help of Kuratowski theorem,we aimed the removing all Kuratowski subgraphs to reach planarity with the minumum number of edge removal.

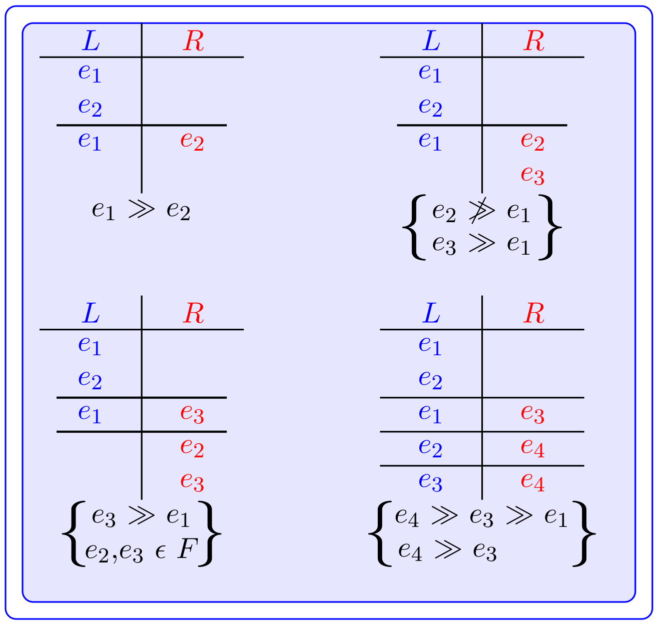


Figure 14: Patterns for Kuratowski in the LR partitioning, see figure 20 for detailed reflections.

3.2 Left right testing for maximal planar subgraph question

The first step for the DFS is the same; see figure 8.

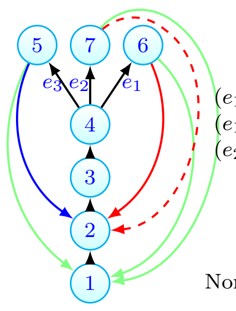
3.2.1 Dividing the left right testing

The LR approach directly focuses on the planarity at the same time of partitioning after the DFS. Because of this reason finding Kuratowski subgraphs is little tricky. After DFS , we do the LR partitioning without testing the planarity. We add the lowpoints to the partitioning table to make the extracting Kuratowski subgraphs, easier. Then instead of testing planarity, we are looking for the patterns that have showed up in the LR partitioning table. For detailed pseudo code see figure 21 .

```
% Use the same DFS as the original LR
Partition($G$)
    $for\ \forall$ fork $v \epsilon\ V$
        $for\ \forall\ (e_i, e_j) \epsilon\ E^+(v)$
            $for\ \forall\ b \epsilon\ B(e_i)$
                $if\ low(b) > lowpoint(e_j)$
                    $if\ flag(e_i) \neq 1$
                        $flag(e_i) = -1; flag(e_j) = 1;$
                        $if\ high(e_1) == v\ then; t = 1; else\ t = 2;$
                        $L(d_1, 1:3) = [v\ b\ t];$
                    $else$
                        $if\ high(e_1) == v\ then; t = 1; else\ t = 2;$
                        $R(d_2, 1:3) = [v\ b\ t];$

% I also added the lowpoints of $e_j$ to corresponding L or R
% if we are able to add an back edge, where I used t=0 to track them.
% Please note that t=1 if $e_i$ is formed of back edges.
```

Figure 15: MPS partitioning procedure

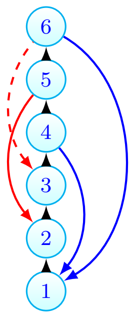


	L	R
(e_1, e_3)	(5,2)	(6,2)
(e_1, e_2)	(7,2)	(6,2)
(e_2, e_3)	(5,2)	(7,2)

	L	R
(e_1, e_3)	(6,1)	(5,1)
	(5,2)	(6,2)
(e_2, e_3)	(6,1)	(7,1)
	(7,2)	(6,2)
(e_1, e_2)	(7,1)	(5,1)
	(5,2)	(7,2)

Non-planar with only one fork

This is why we have fixed the same ends for only if cotree edges involved



L	R
(5,2)	
(6,3)	
(5,2)	(6,3)

Processing all cotree edges

L	R
(4,1)	(6,1),(5,2)
(5,2)	
(6,3)	
(6.1)	(5,2)
(5,2)	(6,3)

Processing the lowpoints

Figure 16: Changes of processing at tables.

3.2.2 Finding all Kuratowski subgraphs with back edges

Now we have the LR table with lowpoints added, and our goal is to find all Kuratowski subgraphs with the an fixed DFS tree. (Changing order of the vertices in the DFS, might result in finding new Kuratowski subgraphs in the graph.) For a detailed pseudo code please see figure 23.

```
% Use the same MPS as the original LR
Extraction($G$)
$for$ $\forall$ $e_1$ $\epsilon$ $L$, (at section $s_i$)
    $for$ $\forall$ $e_2$ $\epsilon$ $L$ (at $s_i$)
        $if$ $\exists e_1 \epsilon$ $L - \{s_i\}$ (at $s_j$)
            $if$ $\exists$ $e_2 \epsilon R$ at $s_j$
                $if$ $(e_1 \gg e_2)$ || $(e_2 \gg e_1)$
                    $C$=[$e_1$, $e_2$, lwpts of $s_i$, lwpts of $s_j$ ]
                    % check if there is repeated vertices at C
                    % check if C is not found before
                $else$ $if$ $\exists$ $e_3 \epsilon R$ at $s_j$
                    $if$ $e_3 \gg e_1$
                        $contradiction$
            $else$
                $for$ $\forall e_3 \epsilon R$ at $s_j$ && $e_3 \gg e_1$
                    $if$ $(e_2, e_3) \epsilon L$( or $R$) at $s_k$
                        $contradiction$
```

Figure 17: MPS extraction procedure

3.2.3 3.2.3 Adding tree edges

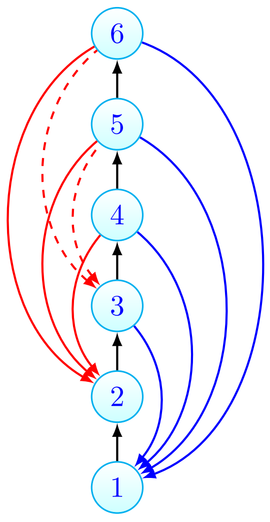
Since there are only two Kuratowski subgraph and all the back edges found in the K_5 or $K_{3,3}$, then only thing left is to sum the number of edges for every vertex that has been showed up in the Kuratowski subgraph. If a tree edge can be removed without losing the Kuratowski subgraph, than we are going to remove that edge. This algorithm is still under revision, so overall results at the program will be only back edges.

3.2.4 Calculating the maximal planar subgraph

After finding all the Kuratowski subgraphs, we are making a matrix (called relevance matrix) , to see which edge has been showed up in which Kuratowski subgraphs.

When we have the relevance matrix, we are to remove an edge, means removing the all Kuratowski subgraphs that that edge has been. When we do this until get the zero matrix as relevance matrix. Then as we do in the DFS, we backtrack to remove all possible edges that makes the relevance matrix is a zero matrix.

After building the partition tables with low-points, we perform the updated LR testing with pattern search. For the testing phase, we are not interested in the low-points; we use them to track down all possible Kuratowski extractions and to store them. It may happen that we find the same Kuratowski subgraphs, or use the same edge again in the same Kuratowski subgraph as a lowpoint. In this case, removing the duplicated information is not difficult. In a stored Kuratowski subgraph, there are 4 low-points (there might be 6 or 8 with an inheritance property) and 2 to 4 back edges (which also can be low-points, as in $K_{3,3}$) so removal of repetitions are not that expensive time-wise. Removal of repeated Kuratowski subgraphs depends on the number of the Kuratowski subgraphs that been discovered.



L	R
(3,1)	(4,1),(5,1) (6,1)
(4,2)	
(5,2)	
(6,2)	
(4,1),(4,2)	(5,1),(6,1)
(5,2)	(4,2)
(6,2)	
(5,3)	
(6,3)	
(5,1),(5,2) (5,3)	(6,1),(6,2)
(6,2)	(5,2)
(6,3)	(5,3)

.

Figure 18: K_6 without (6,4),with the idea of sectioning and lowpoint.

These are the Kuratowski extractions from K_6-(6,4);

(3,1) (4,1) (6,1) (4,2) (5,2) (5,3)

(3,1) (4,1) (6,1) (4,2) (6,2) (5,3)

(3,1) (4,1) (5,1) (4,2) (5,2) (5,3)

(3,1) (4,1) (5,1) (4,2) (6,2) (5,3)

(3,1) (4,1) (5,1) (4,2) (5,2) (6,3)

(3,1) (4,1) (6,1) (4,2) (6,2) (6,3)

(3,1) (5,1) (6,1) (5,2) (6,2) (6,3)

(4,1) (6,1) (5,2) (6,3)

(3,1) (5,1) (4,2) (6,2) (6,3)

	Contradictions																				
	1	2	3	4	5	6	7	8	9				1	2	3	4	5	6	7	8	9
(6,1)	1	1	0	0	0	1	1	1	0				0	0	0	0	0	0	0	0	0
(5,1)	0	0	1	1	1	0	1	0	1				0	0	1	1	1	0	0	0	1
(4,1)	1	1	1	1	1	1	0	1	0				0	0	1	1	1	0	0	0	0
(3,1)	1	1	1	1	1	1	1	0	1				0	0	1	1	1	0	0	0	1
(6,2)	0	1	0	1	0	1	1	0	1				0	0	0	1	0	0	0	0	1
(5,2)	1	0	1	0	1	0	1	1	0				0	0	1	0	1	0	0	0	0
(4,2)	1	1	1	1	1	1	0	0	1				0	0	1	1	1	0	0	0	1
(6,3)	0	0	0	0	1	1	1	1	1				0	0	0	0	1	0	0	0	1
(5,3)	1	1	1	1	0	0	0	0	0				0	0	1	1	0	0	0	0	0

Figure 19: This is deleting the edge (6,1) from relevance matrix.

As we see in the second part of Figure 19, removal of (6,1) lefts 3rd, 4th, 5th and 9th coloumns non-zero only, means they are still causing contradictions.

Since there are only 4 Kuratowski subgraphs left out of original 9 Kuratowski subgraphs, deleting any of (5,1),(3,1), (4,2) makes the graph(K_6-(6,4)) planar can easily be seen. Also, deleting the sets of (6,2),(5,2) and (6,3),(5,3) also makes the graph planar. In this case, with deleting (6,1), one of 3 edges, or one of the two sets, makes the graph planar, so we have three MPS, and 2 maximal planar subgraph. (Note: For K_6, the minimum number of edges to be removed to get a planar graph is 3.) (Note 2: I have dashed (6,3) and (5,3) in the figure such that removing them makes the graph planar; and if one looks at the relevance matrix, the values of (6,3) and (5,3) complement each other.)

Chapter 4

COMPUTATIONAL COMPLEXITY

4.1 Tremaux tree complexity

Tremaux tree, also known as DFS tree, algorithm has been published in a linear time algorithm in the number of edges. We have not change or add anything on this issue. So, this part is still linear.

4.2 Left Right table complexity

While the LR testing is linear on the number of edges, this part is also linear. Also, adding the lowpoints(they are discovered at the DFS) has still linear timing.

4.3 LR testing and pattern search complexity

In this algorithm, we are looking for the edges that appears in the same part of the partition and comparing them to the other parts of partition.

For the first pattern, e_i, there can be maximum of $n-2$ number of edges in the same part of partition, and for the worst case we need to compare them in the another part of the partition, With this we get the linear timing. It is also the essantial of the original LR testing.

For the second pattern, we are looking if the first pattern fails, and looking for another edge in the same part of the second edge, so it is still linear (because, we have this condition means, we have not look for the worst case scenerio in the first pattern, means still linear.)

For the third pattern, this means we could not locate the second edge that we are looking for and found the third edge that greater than first edge, means we need to look to the table again to see if second and third edges appears in another part of the partition. This makes the algorithm quadritic, in the worst case scenerio.

For the forth pattern, this means we have the third and forth edges, that has been appeared in the partition, finding them seperatly has linear timing, but looking for if the third and forth edge is in the same part of the partition, makes the timing quadratic again.

While there are reflections in the patterns, at the worst case scenerio, timing is O(m^2) on the number of edges.

4.4 Adding tree edge complexity

This section is under revision, but in the theory, it is linear on the number of tree edges, since we are to count and compare it to the Kuratowski subgraphs.

4.5 Calculating the maximal planar subgraph complexity

The relevance matrix has been made of number of edges and number of Kuratowski subgraphs. Since number of Kuratowski subgraphs is related to density of a graph, while we can calculate how many Kuratowski subgraphs we will have bedfore run the program, we can approximate the number of output with the density(D) of program.

Chapter 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In this work, we tried to fill an important gap in the Left-Right approach and its corresponding algorithm. We have added a new procedure to original linear time algorithm without losing the linear timing.

In addition to deciding the planarity with the updated version of the left right algorithm, we have noticed that without the cost of extra time, we can divide the approach to two parts. At first part, we do the partitioning, where puting all the edges to the left side or the right. Then we can do the left right planarity testing with same efficiency comparing the worst case scenerio of the original algorithm. By this, we were able to extract the Kuratowski subgraphs more efficiently because they show up in the partition in four main patterns and their reflections with the fixed DFS tree. Since DFS tree itself never causes a contradiction, we get all the crossings in the form of back edges. By constracting the relevance matrix and removing the rows by one by until having the zero relevance matrix, we are getting a Maximal Planar Subgraph. This approach may also result in a maximum planar subgraph, but we need to all combinations of DFS tree to get a definite answer.

5.2 Future work

Since, we get this approach to find MPS while looking for a efficient way for two-planarity question(removing vertices one plane to other such that both graphs in the both planes should be planar) our first goal is to find a efficient way to solve that question by using updated LR approach.

Also, we think that this program can get faster results with the help of some heuristic approachs in the relevance matrix step.

if e_1,$e_2\epsilon$ P1L,	$e_1\epsilon$ P2L,	$e_2\epsilon$ P2R,	$e_2 \gg e_1$ or $e_1 \gg e_2$	
if e_1,$e_2\epsilon$ P1L,	$e_1\epsilon$ P2R,	$e_2\epsilon$ P2L,	$e_2 \gg e_1$ or $e_1 \gg e_2$	
if e_1,$e_2\epsilon$ P1R,	$e_1\epsilon$ P2L,	$e_2\epsilon$ P2R,	$e_2 \gg e_1$ or $e_1 \gg e_2$	
if e_1,$e_2\epsilon$ P1R,	$e_1\epsilon$ P2R,	$e_2\epsilon$ P2L,	$e_2 \gg e_1$ or $e_1 \gg e_2$	
if e_1,$e_2\epsilon$ P1L,	$e_1\epsilon$ P2L,	$e_2, e_3\epsilon$ P2R,	$e_3 \gg e_1$ & $e_2 \not\gg e_1$	
if e_1,$e_2\epsilon$ P1L,	$e_1\epsilon$ P2R,	$e_2, e_3\epsilon$ P2L,	$e_3 \gg e_1$ & $e_2 \not\gg e_1$	
if e_1,$e_2\epsilon$ P1R,	$e_1\epsilon$ P2L,	$e_2, e_3\epsilon$ P2R,	$e_3 \gg e_1$ & $e_2 \not\gg e_1$	
if e_1,$e_2\epsilon$ P1R,	$e_1\epsilon$ P2R,	$e_2, e_3\epsilon$ P2L,	$e_3 \gg e_1$ & $e_2 \not\gg e_1$	
if e_1,$e_2\epsilon$ P1L,	$e_1, e_3\epsilon$ P2L,	$e_2\epsilon$ P2R,	$e_3 \gg e_2$ & $e_1 \not\gg e_2$	
if e_1,$e_2\epsilon$ P1L,	$e_1, e_3\epsilon$ P2R,	$e_2\epsilon$ P2L,	$e_3 \gg e_2$ & $e_1 \not\gg e_2$	
if e_1,$e_2\epsilon$ P1R,	$e_1, e_3\epsilon$ P2L,	$e_2\epsilon$ P2R,	$e_3 \gg e_2$ & $e_1 \not\gg e_2$	
if e_1,$e_2\epsilon$ P1R,	$e_1, e_3\epsilon$ P2R,	$e_2\epsilon$ P2L,	$e_3 \gg e_2$ & $e_1 \not\gg e_2$	
if e_1,$e_2\epsilon$ P1L,	$e_1\epsilon$ P2L,	$e_3\epsilon$ P2R,	$e_2, e_3\epsilon P3L$,	$e_3 \gg e_1$ & $e_2, e_3\epsilon F$
if e_1,$e_2\epsilon$ P1L,	$e_1\epsilon$ P2L,	$e_3\epsilon$ P2R,	$e_2, e_3\epsilon P3R$,	$e_3 \gg e_1$ & $e_2, e_3\epsilon F$
if e_1,$e_2\epsilon$ P1L,	$e_1\epsilon$ P2R,	$e_3\epsilon$ P2L,	$e_2, e_3\epsilon P3L$,	$e_3 \gg e_1$ & $e_2, e_3\epsilon F$
if e_1,$e_2\epsilon$ P1L,	$e_1\epsilon$ P2R,	$e_3\epsilon$ P2L,	$e_2, e_3\epsilon P3R$,	$e_3 \gg e_1$ & $e_2, e_3\epsilon F$
if e_1,$e_2\epsilon$ P1R,	$e_1\epsilon$ P2L,	$e_3\epsilon$ P2R,	$e_2, e_3\epsilon P3L$,	$e_3 \gg e_1$ & $e_2, e_3\epsilon F$
if e_1,$e_2\epsilon$ P1R,	$e_1\epsilon$ P2L,	$e_3\epsilon$ P2R,	$e_2, e_3\epsilon P3R$,	$e_3 \gg e_1$ & $e_2, e_3\epsilon F$
if e_1,$e_2\epsilon$ P1R,	$e_1\epsilon$ P2R,	$e_3\epsilon$ P2L,	$e_2, e_3\epsilon P3L$,	$e_3 \gg e_1$ & $e_2, e_3\epsilon F$
if e_1,$e_2\epsilon$ P1R,	$e_1\epsilon$ P2R,	$e_3\epsilon$ P2L,	$e_2, e_3\epsilon P3R$,	$e_3 \gg e_1$ & $e_2, e_3\epsilon F$
if e_1,$e_2\epsilon$ P1L,	$e_1\epsilon$ P2L,	$e_3\epsilon$ P2R,	$e_2, e_3\epsilon P3L$,$e_4\epsilon P3R$,	$e_4 \gg e_3$ & $e_3 \gg e_1$
if e_1,$e_2\epsilon$ P1L,	$e_1\epsilon$ P2L,	$e_3\epsilon$ P2R,	$e_2, e_3\epsilon P3R$,$e_4\epsilon P3L$,	$e_4 \gg e_3$ & $e_3 \gg e_1$
if e_1,$e_2\epsilon$ P1L,	$e_1\epsilon$ P2R,	$e_3\epsilon$ P2L,	$e_2, e_3\epsilon P3L$,$e_4\epsilon P3R$,	$e_4 \gg e_3$ & $e_3 \gg e_1$
if e_1,$e_2\epsilon$ P1L,	$e_1\epsilon$ P2R,	$e_3\epsilon$ P2L,	$e_2, e_3\epsilon P3R$,$e_4\epsilon P3L$,	$e_4 \gg e_3$ & $e_3 \gg e_1$
if e_1,$e_2\epsilon$ P1R,	$e_1\epsilon$ P2L,	$e_3\epsilon$ P2R,	$e_2, e_3\epsilon P3L$,$e_4\epsilon P3R$,	$e_4 \gg e_3$ & $e_3 \gg e_1$
if e_1,$e_2\epsilon$ P1R,	$e_1\epsilon$ P2L,	$e_3\epsilon$ P2R,	$e_2, e_3\epsilon P3R$,$e_4\epsilon P3L$,	$e_4 \gg e_3$ & $e_3 \gg e_1$
if e_1,$e_2\epsilon$ P1R,	$e_1\epsilon$ P2R,	$e_3\epsilon$ P2L,	$e_2, e_3\epsilon P3L$,$e_4\epsilon P3R$,	$e_4 \gg e_3$ & $e_3 \gg e_1$
if e_1,$e_2\epsilon$ P1R,	$e_1\epsilon$ P2R,	$e_3\epsilon$ P2L,	$e_2, e_3\epsilon P3R$,$e_4\epsilon P3L$,	$e_4 \gg e_3$ & $e_3 \gg e_1$

Figure 20: Reflections of patterns

In the LR partitioning: $e_1\epsilon$P1L means, left side of the first part that we are looking for, P2 means the any part after P1, and P3 means the any part after P2

```
for ∀fork f
    for $\forall\ e, f \epsilon E^{+}(f)$
        s1:= return edges of e
        s2:= return edges of f
        lwp1:= lowestpoint of e
        lwp2:= lowestpoint of f
        for i=1:size(s1)
                if s1(i)<f && lwp1<s1(i)
                    if flag1==1; putinto L; do delays; flag2=2;
                    elseif flag1==2; putinto R; do delays; flag2=1;
                    elseif s1(i)ϵL ; flag1=1; flag2=2; putinto L; do delays;
                    elseif s1(i)ϵR ; flag1=2; flag2=1; putinto R; do delays;
                    else putinto delays1

        for i=1:size(s2)
                if s2(i)<f && lwp2<s1(2)
                    if flag2==1; putinto L; flag1=2;do delays;
                    elseif flag2==2; putinto R; flag1=1; do delays;
                    elseif s2(i)ϵL ; flag1=2; flag2=1; putinto L; do delays;
                    elseif s2(i)ϵR ; flag1=1; flag2=2; putinto R; do delays;
                    elseif i==size(s2); putinto R; do delays; % worst case
                    else putinto delays2
```
.

Figure 21: Partitioning procedure

```
function delays

if flag1==1
    for i=1:size(delays1); putinto L;
else
    for i=1:size(delays1); putinto R;

if flag2==1
    for i=1:size(delays2);putinto L;
else
    for i=1:size(delays2);putinto R;
```

Figure 22: Delays function

```
function PatternSearch
    do Pattern1(L,L,R)
    do Pattern1(L,R,L)
    do Pattern1(R,L,R)
    do Pattern1(R,R,L)



function Pattern1(A,C,D)
    for i=1:size(A)
        if A(i,4)==0; do lowpoints;% since there can be more than one lowpoints, consider them
        if A(i,4)>0
            $e_1$=A(i,1:2);
            s:= other edges in this part, called P1;
            while(j<size(s)
                $e_1$=A(s(j),1:2);
                for k=1:size(C)
                    if $e_1 \epsilon$ C
                        for k=1:size(D)
                            if $e_2 \epsilon$D
                                if $e_2 \gg e_1$
                                    contradiction
                            do Pattern2(A,C,D)
                            do Pattern3(A,C,D)
```
.

Figure 23: Kuratowski extraction

```
function Pattern2(A,C,D)
    if $e_2 \not\gg e_1$ & $e_2 \epsilon$P2D
        for t=1:size(D)
            if D(t,4)>0;$e_3$=D(t,1:2);
                if $e_3 \gg e_1$
                    contradiction
        for t=1:size(C)
            if C(t,4)>0;$e_3$=C(t,1:2);
                if $e_3 \gg e_2$
                    contradiction


function Pattern3(A,C,D)
    if $e_2 \not\epsilon$P2D
        for t=1:size(D)
            if D(i,4)>0; $e_3$= D(t,1:2);
                if $e_3 \gg e_1$
                    for i=1:size(C)
                        if $e_2 \epsilon$C; flag1=1;
                        if $e_3 \epsilon$C;flag2=1;
                        if flag1==1 && flag2==1
                            if C(i,4)==2
                                contradiction
                            else
                                do pattern4(A,D)
                    for i=1:size(D)
                        if $e_2 \epsilon$D; flag1=1;
                        if $e_3 \epsilon$D;flag2=1;
                        if flag1==1 && flag2==1
                            if D(i,4)==2
                                contradiction
                            else
                                do pattern4(A,C)
```
.

Figure 24: Pattern2 and Pattern3 algorithms

```
function Pattern4(A,X)
    for i=1:size(X)
        if X(i,4)>0
            if X(i,3)==P3
                if $e_4 \gg e_3$
                    contradiction
```

For e_i=(u_i, v_i) and e_j=(u_j, v_j) , $e_i \gg e_j$ means $u_i > u_j$ and $v_i > v_j$

.

Figure 25: Pattern4 algorithm

Chapter 6

APPENDIX

Theorem 1: For a given graph G, and the partition of LR , every pattern of ours ends up a Kuratowski subgraph.

Proof:

Case 1: For the first pattern, we need two edges in one section at the same side and in the other section in the different parts, where $e_1 >> e_2$ or they are not comparable.

if $e_1, e_2 \epsilon S_i$ then there has to a return edge (u_1, v_1) that ends lower than $low(e_1), low(e_2)$ and that return edge also should be an back edge, means $u_1 := fork$. Now from this section we know that,where $e_1 = (x_1, y_1), e_2 = (x_2, y_2)$

$\{x_1, x_2\} > u_1 > \{y_1, y_2\} > v_1$ (1)

For second section, $e_1 >> e_2$ means $x_1 > x_2$ and $y_1 > y_2$. And with the assumption of e_2 is the lowest return point of back edges for e_1 to be part of that section, then we also need a lowest point of back edges for e_2 to be part of that section, call (u_2, v_2) where $y_2 > v_2$.With the (1) equation we get;

$x_1 > x_2 > u_1 > y_1 > y_2 > \{v_1, v_2\}$ (2)

with the addtion of DFS tree to (2), we have a Kuratowski subgraph.

If they are not comparable, then x_1 and x_2 must be in the different brances at DFS after second fork f.

If they have the same endpoints, then there has to be two return edges as lowpoints, so they both can be at same section. (u_3, v_3) and (u_4, v_4) where $f > y_1 = y_2 > \{v_3, v_4\}$. With the (1) equation, we get,

$\{(x_1, u_3), (x_2, u_4)\} > f > u_1 > y_1 = y_2 > \{v_1, v_3, v_4\}$ (3)

with the addtion of DFS tree, we have a Kuratowski subgraph.

If they have different endpoints, assume $y_1 > y_2$, then e_2 can be a return edge for e_1 to be in that section, and we need only one return edge that can be a lowpoint for e_2.Call that return edge (u_5, v_5) where x_1 and u_5 has same direction after fork f, and $y_2 > v_5$. With the help of (1) equation, we get;

$\{(x_1, u_5), x_2\} > f > u_1 > y_1 > y_2 > \{v_1, v_5\}$ (4)

Since we also get a Kuratowski subgraph at equation (4), then our first pattern results in only Kuratowski subgraph. The other patterns has the same idea behind it, so we will skip it for.

Theorem 2: Every contradiction at the LR algorithm yields one of our patterns.

Proof :

We will use the characteristics of DFS tree to get a contradiction. If we have a contradiction for edge e_1, it means e_1 is discovered at both left side and right side at two different section. To have this kind of property, there are e_2 with the same side and different side with the e_1 in the both section, there are e_2 at one section at opposite side and e_3 is at the other section with the opposite side, with no option of switching sides, means e_2 and e_3 is also at opposite sides. If dont get a contradiction, it means the graph is planar. We added a table for corresponding graphs.

Case 1: Consider we have a Type[iii] characteristic as a subgraph. Then we have 4 back edges, e_1, e_1' at one side and e_2, e_2' at other side.

Case 1.1 If we have another Type[iii] characteristic with e_1, we have tree options.Where e_1 and e_3 at different sides.

Case 1.1.1 Second fork can be after first fork, which implies e_2 and e_3 can not be comparable. This implies that , e_1 and e_3 is at the same side for first fork, while e_2 in the different side, where for the second fork, e_1 and e_3 are at different sides. This means that it is a contradiction and, without e_2 , it is one of our patterns.

Case 1.1.2 There could be three brancing at first fork. This means, for the first couple e_1 and e_2 at different sides, for second couple e_1 and e_3 at different sides, for the third couple e_2 and e_3 at different sides, it means we have a contradiction, and it is also one of our patterns.

Case 1.1.3 Second fork can be before first fork. Since second fork comes before first fork, if we do comparision, e_1 and e_2 at the same side, while e_3 at the other side for send fork. And since e_1 and e_2 is also different sides for first fork, it implies that we have a contradiction, and without e_3, it is one of our patterns.

Case 1.2 If we have another Type[ii] .

Case 1.2.1 Assume e_1 and e_3, e_3' to be different sides. Than it means $\{x_3, x_3'\} > x_1$ and $y_3 > y_1 > y_3'$

Case 1.2.1.1 if $y_3 \geq f_1$, if $y_3' \leq x_1'$ than we have no contradiction, Case 1.2.1.2 if $y_3 \geq f_1$ and $y_3' > x_1'$ than for the first fork, e_1 and e_3' are at the same side while e_2 on the other side. For the x_1, e_3 and e_3' are the same side while e_1 in the other side. Without e_2 and e_3 this yields a contradiction and it is oone of our patterns.

Case 1.2.1.3 if $y_3 < f_1$, for the first fork, e_1 and e_3 are the same side while e_2 at other and for the x_1 , e_1 and e_3 are at different sides, it yields a contradiction, and without e_2 it is one of our patterns.

Case 1.2.2 Assume e_1 and e_3 to be different sides and $\{x_1, x_1'\} > x_3$ and $y_1 > y_3 > y_1'$

Case 1.2.2.1 if $x_3 > f_1$, for the first fork, we have e_1 and e_3 at the same side, while e_2 at other side, and for the x_3, we have e_1 and e_3 are at the different sides, it yields a contradiction, and without e_2 it is one of our patterns.

Case 1.2.2.2 if $x_3 = f_1$, it yields a similar result for case 1.1.2.

Case 1.2.2.3 if $x_3 < f_1$, it also have a similar result with case 1.1.3.

Case 1.3 if we have Type[i] for e_1 and e_3 to be same side. Since there are quite a few case, and they end up in similar results, we are not going to show each of them.

Case 1.4 if we have Type[iv] for e_1 and e_3 . Then there is three cases,

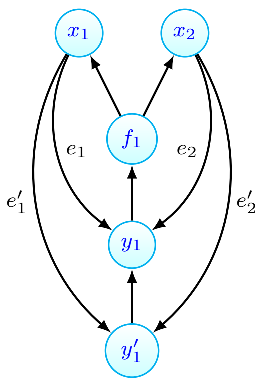
Case 1.4.1 if $x_3 > f_1$ which is not a contradiction.

Case 1.4.2 if $x_3 = f_1$ which is not a contradiction.

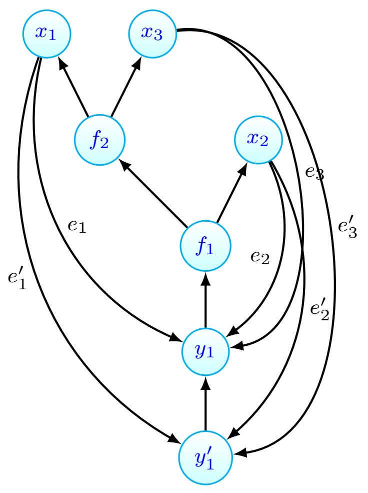
Case 1.4.3 if $x_3 < f_1$ which yields a contradiction with using e_1 and e_2.

For Case 2, we have choosen Type[ii], with subcases, for Case 3 we have choosen Type [iii] with subcases, and finally for Case[iv] we have choosen Type [iv] with subcases. Since they are all results in one of our patterns or planarity, one can use this steps to go further in the proof easily.

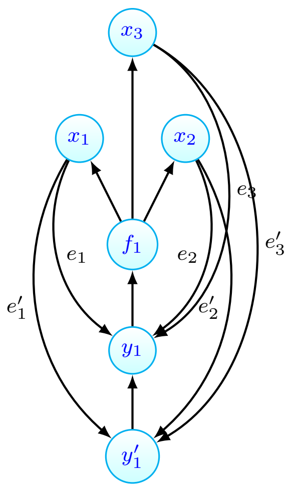
.



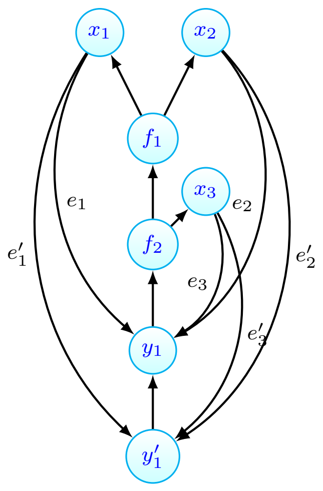
(Case 1)
[$y_1 = y_2$] and [$y_1' = y_2'$]



(Case 1.1.1)



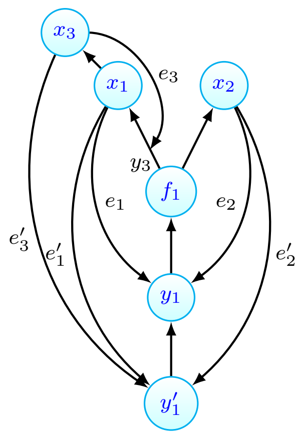
(Case 1.1.2)



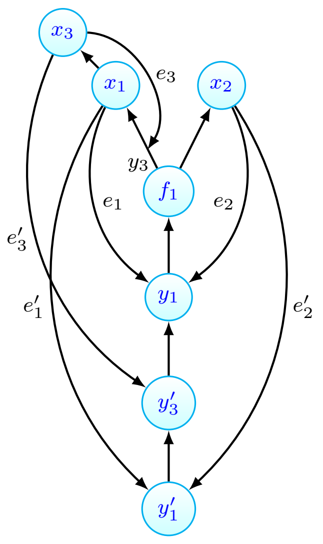
(Case 1.1.3)

Figure 26: Adding a Type[iii] to a Type[iii]

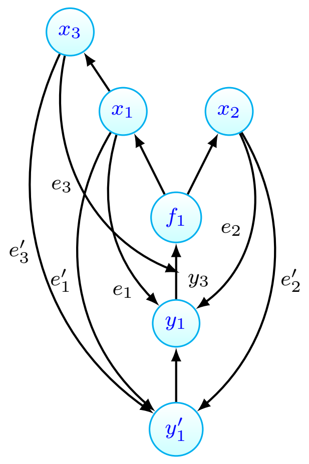
.



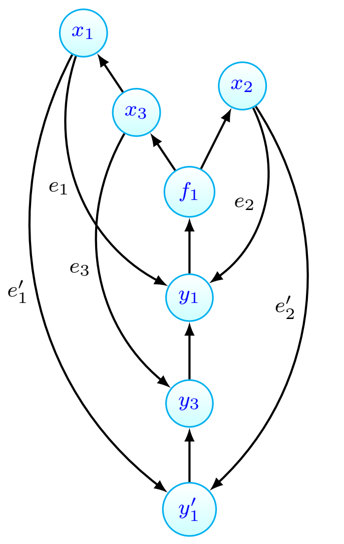
(Case 1.2.1.1)



(Case 1.2.1.2)



(Case 1.2.1.3)



(Case 1.2.2)

Figure 27: Adding a Type[ii] to a Type[iii]

Theorem : If there is Kuratowski Subgraph, then we can discover it by our algorithm.

Proof:

Assume there is a Kuratowski subgraph that our algorithm did not discover with back edges, it means the graph G is planar because we are using a planarity testing algorithm. But by Kuratowski's theorem, a graph is planar if and only if there is no subgraph that subdivsion of Kuratowski subgraph. It is a contradiction.

Assume there is a Kuratowski subgraph that has no back edges discovered at DFS procedure. It means, there is no cycle at Kuratowski subgraph, which is again a contradiction.

Bibliography

[1] Boyer, J., & Myrvold, W. (2004).On the Cutting Edge: Simplified $O(n)$ planarity by edge addition. *Journal of Graph Algorithms and Applications,* 8(3), 241-273.

[2] Boyer, J., Cortese, P., Patrignani, M., & Di Battista, G. (2003). Stop minding your P's and Q's: implementing a fast and simple DFS-based planarity testing and embedding algorithm. *International Symposium on Graph Drawing*, 25-36.

[3] Chimani, M., Klein, K., & Wiedera, T. (2016). A note on the practicality of maximal planar subgraph algorithms. Appears in the Proceedings of the 24th International Symposium on Graph Drawing and Network Visualization(GD 2016), (p. 1609.02443).

[4] Chimani, M., Mutzel, P.,& Schmidt, J. (2007). Efficient extraction of multiple Kuratowski subdivisions. Graph Drawing, 15th International Symposium, Sydney, Australia.

[5] de Fraysseix, H., & de Mendez, P. (2002). Pigale: Public implementation of a graph algorithm library and editor. *Software project at pigale.sourforge.net (GPL licence)*, 599-620.

[6] de Fraysseix, H., & de Mendez, P. (2003). On cotree-critical and DFS cotree-critical graphs. *J. Graph Algorithms Appl.*, 7(4), 411-427.

[7] de Fraysseix, H., & de Mendez, P. (2012). Tremaux trees and planarity. *European Journal of Combinatorics*, 33, 279-293.

[8] de Fraysseix, H., & Ossona de Mendez, P. (2001). A characterization of DFS cotree critical graphs. International Symposium on Graph Drawing, 84-95.

[9] de Fraysseix, H., & Rosenstiehl, P. (1982). A depth-first-search characterization of planarity. *Annals of Discrete Mathematics*, 13, 75-80.

[10] de Fraysseix, H., & Rosenstiehl, P. (1983). Système de reference de Tremaux D'une representation plane D'Un graphe planaire. *North-Holland Mathematics Studies*, 293-302.

[11] de Fraysseix, H., & Rosenstiehl, P. (1985). A characterization of planar graphs by trémaux orders. *Combinatorica*, 5(2), 127-135.

[12] de Fraysseix, H., de Mendez, P., & Rosenstiehl, P. (2006). Depth-first search and planarity. *Int. J. Found. Comput. Sci.*, 17(5), 1017-1030.

[13] Djidjev, H. (1984). On some properties of nonplanar graphs. *C.R. Acad. Bulgare Sci.*, 37, 1183-1184.

[14] Djidjev, H. (2006). A linear-time algorithm for finding a maximal planar subgraph. *Siam J. Discrete Math.*, 20(2), 444-462.

[15] Even, S., & Tarjan, R. (1976). Computing an st-numbering. *Theoretical Qmputer Science*, 2, 339-344.

[16] Garey, M., & Johnson, S. (1979). Computers and Intractability: A guide to the theory of NP-completeness. *Freeman&Co.* San francisco.

[17] Hedtke, I. (2017). Minimum genus and maximum planar subgraph: Exact algorithms and general limits of approximation algorithms. Ph.D. Thesis Osnabrück University. Retrieved from repositorium.ub.uos.de/handle/urn:nbn:de:gbv:700-2017082416212.

[18] Hopcroft, J., & Tarjan, R. (1974). Efficient planarity testing. *Journal of the Association for Computing Machinery,* 21(4), 549-568.

[19] Jugner, M., & Mutzel, P. (1996). Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica*, 16, 33-59.

[20] Lempel, A., Even, S., & Cederbaum, I. (1967). An algorithm for planarity testing of graphs. *In: Rosenstiehl, P. , Theory of Graphs,* Gordon and Breach, 215-232.

[21] Marek-Sadowska, M. (1979). Planarization algorithm for integrated circuits engineering. In Proceedings of the IEEE International Symposium on Circuits and Systems (pp. 919-923). Piscataway,NJ: IEEE Press.

[22] Mehlhorn, K., & Mutzel, P. (1996). On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. *Algorithmica*, 16(2), 233-242.

[23] Mehlhorn, K., & Näher, S. (1995). LEDA: A library of efficient data types and algorithms. *Communications of the ACM*, 38(1), 96-102.

[24] Mehlhorn, K., Mutzel, P., & Näher, S. (1993). An implementation of the Hopcroft and Tarjan planarity test and embedding algorithm. *Technical Report MPI-I* Max-Planck-Institut für Informatik, Saarbrücken, 93-151.

[25] Pasedach, K. (1976). Criterion and algorithms for determination of bipartite subgraphs and their application to planarization of graphs. *in Graphen-Sprachen und Algorithmen in Graphen*, 175-183.

[26] Wu, W. (1955). On the realization of complexes in Euclidean space I. *Acta Mathematica Sinica*, 5, 505-552.

[27] Tamassia, R., G. Di Battista, and C. Batini(1988). Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man and Cybernetics*, 18, 61-79.

[28] Thomas Lengauer(1990). Combinatorial Algorithms for Integrated Circuit Layout.(pp. 7-8) Hoboken,NJ: John Wiley & Sons Ltd.