WEAKLY TRIANGULATED GRAPHS

A Project Report Submitted

for the Course

CS498 Project I

by

Charanjit Singh Ghai

(Roll No. 11010178)

Rachuri Anirudh

(Roll No. 11010155)

Sparsh Kumar Sinha

(Roll No. 11010166)

to the

DEPARTMENT OF CSE

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

GUWAHATI - 781039, INDIA

November 2014

CERTIFICATE

This is to certify that the work contained in this project report entitled "**Weakly Triangulated Graphs**" submitted by **Charanjit Singh Ghai** (**Roll No.: 11010178**), **Rachuri Anirudh** (**Roll No.: 11010155**) and **Sparsh Kumar Sinha** (**Roll No.: 11010166**) to Department of Computer Science and Engineering, Indian Institute of Technology Guwahati towards the requirement of the course **CS498 Project I** has been carried out by them under my supervision.

Guwahati - 781 039	(Dr. S. V. Rao)
November 2013	Project Supervisor

ABSTRACT

A graph G = (V, E) is said to be triangulated (also called chordal) if it has no chordless cycle of length four or more. Such a graph is said to be rigid if, for a *valid* assignment of edge lengths, it has a unique linear layout and non-rigid otherwise. An assignment of lengths to the edges of G is said to be valid if the distances between the adjacent vertices in a linear placement is consistent with the lengths assigned to the edges of G. A graph G = (V, E) is weakly triangulated if neither G nor its complement $\overline{G}$ contains a chordless cycle of five or more vertices. In this report, we shall be looking into the problem of generating random weakly triangulated graphs and finding linear layouts of chordal bipartite graphs which is a sub-class of weakly triangulated graphs.

Contents

List of Figures

Chapter 1

Introduction

Point Placement Problem is a problem where we are given distances between all possible pairs of n nodes, and we have to find a possible placement of those nodes in space such that those distances are not violated. This problem holds significant importance in the field of molecular biology, where given the inter-atomic distances for all possible pairs of atoms in a molecule, we have to come up with an arrangement of these atoms in the three dimensional space so that the distance constraints are not violated. This problem can be visualised as a graph where there is a vertex corresponding to each node(atom) and an edge for a distance constraint with weight equal to the distance value. Here by absence of edge we signify absence of the distance constraint between the corresponding two atoms or vertices. It has been shown that the point placement problem in general (even in two dimensions) is NP-complete. Thus, a simplified and similar problem which is point placement for weakly triangulated graphs where the aim is to place these nodes linearly (i.e. in one dimension) is studied in detail. Also, finding valid edge assignments for getting linear layouts etc. are similar problems which are studied in the domain of weakly triangulated graphs. We discuss two problems in the following sections falling under the same domain.

1.1 Problem Definitions

1.1.1 Generate Random Weakly Triangulated Graphs

Owing to proposal of algorithms for computing linear layouts of weakly triangulated graphs, and other algorithms solving various problems related to weakly triangulated graphs, there is a need for generating these graphs randomly. Thus, we shall be looking into how to generate these graphs randomly. Precisely, the aim here is to come up with an algorithm which can generate random weakly triangulated graphs. Further, the probability of generation of each such graph should be non zero.

1.1.2 Chordal Bipartite Graphs

A chordal bipartite graph is one which is a bipartite graph as well as weakly triangulated. [1] showed how to compute all linear layouts of a weakly triangulated graph, for a valid assignment of lengths to the edges of the graph. Our aim is to extend this result to come up with an algorithm to solve this problem efficiently for chordal bipartite graphs, a sub-class of the weakly triangulated graphs using their properties.

Chapter 2

Generating Random Weakly Triangulated Graphs

2.1 Related Work

Although our work was related to graph generation, we did a survey on generating random trees. This choice would not need any justification after reading the Preliminaries section. The survey by **Sean Luke and Liviu Panait** [3] gave many algorithms for generating random trees constrained by depth of the tree. The main idea revolves around generating a node as a terminal(leaf) or a non terminal(internal node) of some degree and recursively calling the procedure of max depth-1 for each child of the non terminal. This tree in some sense can be related with the parse tree of mathematical expressions. Also, **Hitoshi IBA** [2] gave an algorithm to generate random trees by generating random strings of x and y. Generation of random string is followed by parsing the string from left to right.Each encountered x is pushed on to a stack (as a leaf node), while for each y with k consecutive leading x nodes, k nodes are popped from the stack and made the children of current (new) node which itself is finally pushed onto stack. The string randomly generated is constrained to have n x and $(n-1)$ y.

2.2 Preliminaries

In this section, we define terms and state results required for the work.

We define a weakly triangulated graph as follows:

Definition 2.2.1. A graph is said to be weakly triangulated if it does not have a chordless cycle of length 5 or more.

An edge of a graph is called a peripheral edge if it is not the middle edge of a P_4 (chordless path of four vertices). The following constructive characterization, analogous to the perfect elimination ordering of a triangulated graph, is central to our approach.

Theorem 2.2.2. *A graph is weakly triangulated if and only if it can be generated in the following manner:*

1. *Start with an empty graph G_0.*

2. *Repeatedly add an edge e_j to the graph G_{j-1} to create the graph G_j such that e_j is a peripheral edge of G_j.*

A total order of the m edges $(e_1, e_2, ..., e_m)$ of a graph G is a peripheral edge order if for $1 \leq j \leq m$, e_j is peripheral in the graph $G_j = (V, E_j)$, where $E_j = \{e_1, e_2, ..., e_j\}$. Thus the following theorem is equivalent to Theorem 1:

Theorem 2.2.3. *A graph is weakly triangulated if and only if it admits a peripheral edge order.*

A graph G is *minimally line rigid* if it has no proper induced *subgraph* that is line rigid. For example, the weakly triangulated graphs $K_{2,3}$ and K_3 are minimally line rigid. A subgraph of a graph G is *maximally line rigid* if it has no proper induced *supergraph* that is line rigid.

A component of G is a maximally connected subgraph. A biconnected component is a 2-connected component. When G is connected, it decomposes into a tree of biconnected components called the block tree of the graph. The blocks are joined to each other at shared vertices called cut vertices or *articulation points*. An edge $\{u, v\}$ of G is a *hinge edge* if removal of the vertices u and v and the edges incident on these disconnects G into three or more disjoint components.

The rigidity structure of G is characterized in the following results.

Lemma 2.2.4. *The only minimally line rigid subgraphs of a weakly triangulated graphs are the* $K_{2,3}$ *and* K_3*.*

Theorem 2.2.5. *The rigidity graph of a weakly triangulated graph without articulation points and hinge edges is a tree.*

2.3 Restricted Random Weakly Triangulated Graph

In this section, we describe an algorithm for generation of weakly triangulated graph without articulation point and hinge edge. The rigidity structure of weakly triangulated graph without articulation point and hinge edges can be represented as a tree with the following properties:

- Each node can be a quadrilateral or a maximally rigid sub-graph of G.
- Each node which corresponds to a quadrilateral can have degree ≤ 4.

• No two nodes both of which correspond to a maximally rigid sub-graph of G can be adjacent to each other in the rigidity tree.

Remark 2.3.1. If we leave aside the problem of labelling each node of the rigidity tree as to a quadrilateral or a maximally rigid sub-graph, the only constraint in the rigidity tree from a structural point of view is that no two nodes with degree ≥ 5 should be adjacent to each other. This constraint is enforced because as soon as the degree of two nodes becomes ≥ 5, both of them have to be labelled as a maximally rigid sub-graph and no two maximally rigid sub-graphs can be adjacent to each other. We refer this as the **Structural Constraint** of the Rigidity Tree.

Taking these constraints into account and the ideas that we learnt for generating random trees, we shall give an algorithm to generate a random n node rigidity tree.

2.3.1 Random Rigidity Tree

We denote by n, k, m respectively the number of nodes in the rigidity tree, number of nodes in the final random weakly triangulated graph and number of edges in the final random weakly triangulated graph. Note that the aim here is not to come up with an algorithm which generates all possible valid rigidity trees of n nodes with equal probability rather, the aim is to come up with an algorithm which generates all possible rigidity trees of n nodes with some non zero probabilities and does not generate anything else. We also thus expect an integer n as an argument to our algorithm.

Our algorithm generates a valid Rigidity tree of n nodes using Recursion. In base case, it returns a tree with only one node when argument n equals one. When n is greater than one, it recursively generates a valid Rigidity tree of $n - 1$ nodes and then builds a set of candidates adjacent to which a new node can be added to get a valid n node Rigidity Tree. The candidate set contains only those nodes which either don't have degree equal to four or those which have degree four but do not have any neighbour which has degree five. The reason such nodes are excluded is that if a node has degree greater than or equal to five, then it has to represent a maximally rigid component, and two nodes which correspond to maximally rigid components can not be adjacent to each other in the Rigidity tree.

Algorithm 1 Generate Random Rigidity Tree(n)

if $n = 1$ **then**
    $G_n \leftarrow$ Tree with only one node
else
    $G_{n-1} \leftarrow$ Generate Random Rigidity Tree(n-1)
    $L \leftarrow$ Set of degree 4 vertives having at least one adjacent vertex with degree ≥ 5
    Add a node adjacent to a random node not in L
    $G_n \leftarrow$ final graph

return G_n

We claim that the above procedure indeed generates a random rigidity tree with n nodes. We shall prove our claim by induction. But, Let us first consider the following Lemma.

2.3.2 Proof of Correctness

Lemma 2.3.2. *Consider generation process of a n node rigidity tree. Each intermediate rigidity tree in this generation process is also a valid rigidity tree.*

Proof. The correctness of the lemma is easy to see as addition of a node can only violate the structural constraint of the rigidity and not removal of a node. Thus, if we have a valid node rigidity tree, then tree obtained after removing any node of the tree must also be a valid rigidity tree because if it was not, then the originial tree will not be a valid rigidity tree thus contradicting the assumption itself. □

Coming back to proof of the correctness of the algorithm, the induction statement P_n is as follows:
P_n: The algorithm generates all rigidity trees of n vertices with some non zero probability and generates no tree which violates the constraints on the rigidity tree.

Proof. Base Case: Clearly P_1 is true since the only rigidity tree possible with one node is a tree containing only one node and no edges. Thus P_1 is trivially true.

Inductive Hypothesis: Let P_n be true for all $n \leq k$.

Inductive Step: When the algorithm is called with $(k+1)$ as parameter, it first calls itself recursively with k as parameter. Now assuming that the P_k is true, we can say that the Algorithm generates all possible rigidity trees with k nodes with some probability and no tree which does not follow the constraints is generated. Now consider any valid $(k+1)$ node rigidity tree and a k node sub-graph of this rigidity tree. Clearly the k node sub-graph is a rigidity tree of k node owing to structural property of the rigidity tree. Also, the left over node will be a leaf node which could either be adjacent to a node with degree $= 4$ which itself has no node with degree ≥ 5 adjacent to it or a node which does not follow these constraints. Since the algorithm considers both the cases, and the only violation to structural property can occur if 2 nodes with degree ≥ 5 become adjacent, we don't violate the structural property while considering all the possibilities for adding a node.

Thus, P_{k+1} is true.

Thus by Mathematical Induction we can say that P_n is true $\forall$ n $\geq$ 1.

$\square$

2.3.3 Assigning Rigidity Labels

Having generated a random Rigidity Tree, we now want to fix the type of each node that is determine which all nodes are Maximally Rigid Components and which all nodes are Quadrilaterals.

Algorithm 2 Assigning Labels to Nodes (Rigidity Tree rgt)

```
S_1 ← Set of nodes in rgt with deg ≥ 5
S ← Set of all nodes in rgt
for node n in S_1 do
    Label[n] ← Maximally Rigid
    S ← S − {n}
    for neighbor s of n do
        Label[s] ← Non-Rigid
        S ← S − {s}
while S ≠ φ do
    Choose any random node n from S
    Choose a random number k
    if k is odd then
        Label[n] ← Maximally Rigid
        S = S − {n}
        for neighbor s of n do
            Label[s] ← Non-Rigid
            S ← S − {s}
    else
        Label[n] ← Non-Rigid
        S = S − {n}
return Label
```

The above algorithm first fixes the label of the nodes which have to Maximally Line Rigid. Once done, it fixes the labels of the immediate neighbors as Quadrilaterals. This is what we term as the *one neighborhood expansion* rule, i.e. if a node is fixed to be Maximally Line Rigid, then fix the label of its' neighbors as Quadrilateral. For the nodes other than those for which the Label was fixed to be Maximally Line Rigid i.e. those which had degree not equal to five, the algorithm picks a random node and assigns a random label and accordingly applies the one neighborhood expansion rule.

2.3.4 Generating Line Rigid Graph

Having generated a random Rigidity tree and fixing the labels of each node randomly, we now wish to generate the random *Maximally Rigid Component* nodes of the rigidity tree. For this we need to figure out an algorithm which can generate random Line Rigid graphs. The Algorithm 3 presented on next page does that.

2.3.5 Proof of Correctness

Clearly, any graph generated by this algorithm is Line Rigid since we mainly try and apply 3 operations on an already generated Rigid Component which are:

1) Adding a new vertex adjacent to at least 2 vertices of the original component. Clearly, if the original component is Line Rigid, this new component will also be line rigid since at least 2 line constraints are applied on the new vertex, hence its' position would be fixed and we would have only one arrangement of vertices satisfying the edge weight constraints. 2) Joining 2 non adjacent vertices of the original component will also lead to a line rigid component if the original component was line rigid. This is easy to see, since the edge weight of this new edge is going to be fixed by the original component itself, because it was line rigid. 3) Superimposing 2 line rigid components. This will also lead to a line rigid component, since the relative

Algorithm 3 Generate Random Line Rigid Graph(initialComponent)

$final \leftarrow \phi$
if $initialComponent \neq \phi$ **then**
    Choose a random number p
    if $p\%2 = 0$ **then**
        $final \leftarrow initialComponent$
else
    Choose a random number k
    if $k\%2 = 0$ **then**
        $final \leftarrow$ Generate Random Line Rigid Graph(K_3)
    else if $k\%2 = 1$ **then**
        $final \leftarrow$ Generate Random Line Rigid Graph($K_{2,3}$)
Choose a random number t
if $t\%3 = 0$ **then**
    Choose random number r s.t. $2 \leq r \leq$ number of vertices in $initialComponent$
    Choose r random vertices from $initialComponent$
    Add a new vertex v s.t. v is adjacent to all these r vertices
    $G \leftarrow$ final graph
    $final \leftarrow$ Generate Random Line Rigid Graph(G)
else if $t\%3 = 1$ **then**
    Choose two random vertices from initialComponent s.t. they don't already have an edge between them
    Add an edge between the two vertices
    $G \leftarrow$ final graph
    $final \leftarrow$ Generate Random Line Rigid Graph(G)
else
    $M \leftarrow$ Generate Random Line Rigid Graph(ϕ)
    Choose a random edge e_1 from initialComponent
    Choose a random edge e_2 from M
    Superimpose e_1 on e_2
    Let G be the final graph
    $final \leftarrow$ Generate Random Line Rigid Graph (G)
return $final$

positions of all the vertices except the vertices involved in superposition are fixed, so we will finaly get only one valid linear arrangement of vertices.

All these points rely on one point now that the initial original component must be line rigid. This, however is easy to see again because the only basic components returned in the algorithm are $K_{2,3}$ and K_3 which themselves are line rigid.

Thus, we have shown that any graph generated by our algorithm is line rigid. Now, we need to prove that all possible line rigid graphs can be generated by our algorithm.

Fact: Each line rigid graph must have a $K_{2,3}$ or a K_3 as a subgraph.

The above fact was proved by [1]. Now, we will prove that any Line Rigid graph which has a $K_{2,3}$ or a K_3 as a subgraph can be generated by our algorithm and thus complete the proof.

Without loss of generality, lets' consider a line rigid graph which has a K_3 as a subgraph. Now, consider a vertex which has at least 2 edges incident to any of the vertices of the K_3.

Case 1: No such vertex exists. Case 1a: The vertex is incident to no other vertex. In this case, we have a dangling edge and hence the original graph can not be line rigid. Case 1b: The connected component of the vertex apart from the K_3 has no vertex incident to the K_3 either. In this case, the vertex is an articulation point, and we are not considering those graphs which have an articulation point or hinge edges. Case 1c: The connected component of the vertex(s) apart from K_3 has at least one vertex incident to the K_3. In this case, the vertex from the connected component which has an edge incident to the K_3 must be adjacent to the vertex s otherwise, the graph will not be a weakly-triangulated graph. Now, this adjacent vertex (adj(s)) can either have an edge incident to the same vertex of K_3 to which s had an edge, or to a different vertex.

Case 1ci:When it has an edge to the same vertex, this vertex of K_3 to which both s and adj(s) have an edge becomes articulation point.

Case 1cj: In case it has an edge to a different vertex, we have a quadrilateral in between, which would itself be a different node in the rigidity tree being adjacent to 2 line rigid components.

Thus, we have shown that there exists a vertex which is adjacent to at least 2 vertices of K_3. Now, this same proof will work for any line rigid graph i.e. at each point, if we want to expand the graph by one vertex, that vertex should be adjacent to at least 2 vertices of the original line rigid graph. Further, once we have added the relevant vertices, we can see the second way of expanding the lien rigid component i.e. adding edges between non-adjacent vertices can be used to obtain the entire graph.

Thus, our Algorithm for genration of line rigid graphs is both sound and complete.

Chapter 3

Linear layouts of Chordal Bipartite Graphs

3.1 Definitions

Definition 3.1.1. A bipartite graph is called Chordal Bipartite if it does not contain any chordless cycle of length > 4. Note that if a bipartite graph does not contain a any chordless cycle of length > 4, even its complement graph does not any chordless cycle of length > 4. In other words, a bipartite graph that is also weakly triangulated is called Chordal Bipartite Graph.

Definition 3.1.2. An edge $e = xy$ of a bipartite graph G = *(X,Y,E)* is *bisimplical* if N(x) + N(y) induces a complete bipartite subgraph of *H*. In other words the neighbourhood of X and neighbourhood of Y induce a complete bipartite subgraph. A graph is said to be complete bipartite if every vertex of first part(out of the two parts in bipartite graph) is connected to every other vertex in the second part and vice versa.

Definition 3.1.3. Consider an ordering of edges $\sigma = [e_1, e_2, ..., e_k]$ for a

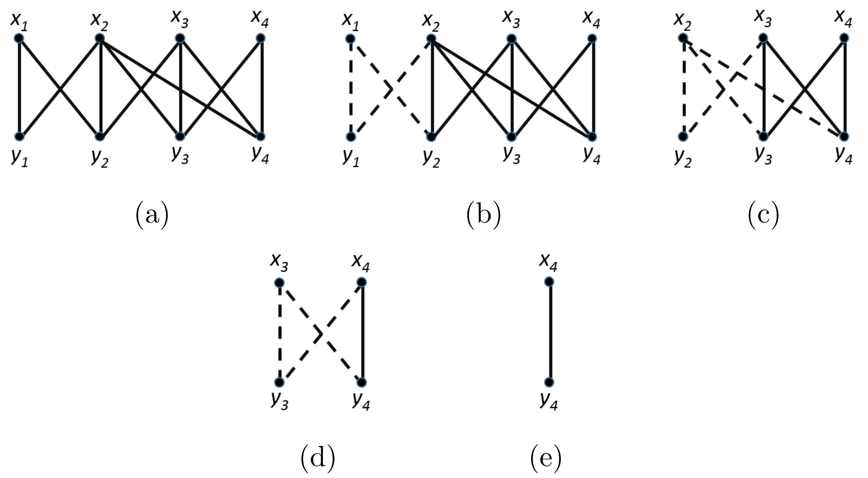


Figure 3.1: Perfect edge without vertex elimination ordering from 3.1a to 3.1e.

chordal bipartite graph G = *(X,Y,E)* with n vertices and m edges. Consider set of graphs $G_0, G_1, ..., G_k$ where $G_0 = G$ and G_i is obtained from G_{i-1} by removing edges xy and the edges incident on x and y and finally G_k is an empty graph. This ordering of edges is known as a *Perfect Edge without Vertex Elimination Ordering*(*PEWVE Ordering*) if each edge e_i in σ is a bisimplical edge for the graph G_{i-1}.

Consider the bipartite graph H as shown in Figure 3.1a. For this graph H, x_1y_1 is a bisimplical edge because (Adj(x_1)={y_2}) + (Adj(y_1)={x_2}) forms a complete bipartite subgraph and x_2y_2 is not a bisimplical edge because (Adj(x_2)={y_1, y_3, y_4}) + (Adj(y_2)={x_1, x_3}) does not form a complete bipartite subgraph as H does not contain edge x_3y_1. From 3.3 we can see that $[x_1y_1, x_2y_2, x_3y_3, x_4y_4]$ is a perfect edge elimination ordering because each of them is bisimplical edge at each stage of perfect edge elimination from 3.1a to 3.1e.

3.2 Computing the Perfect Edge without Vertex Elimination Ordering

According to Klask and Kratsch[4] the Perfect Edge without Vertex Elimination ordering for a graph can easily be calculated from the doubly lexical ordering of the bipartite adjacency matrix in $O(min(m \log n, n^2))$ where m is the number of edges and n is the number of vertices of the graph.

3.3 Perfect Edge without Vertex Elimination As a Subsequence of Peripheral Edge Order

3.3.1 Proposed Algorithm

Lemma 3.3.1. *We can get the Peripheral Edge Order from Perfect Edge Without Vertex Elimination Ordering .*

The algorithm that will described below is applied for Chordal Bipartite graph $G = (X, Y, E)$ with n vertices and m edges.

Description

The algorithm is as follows. Add every edge in the reverse of the PEWVE Ordering after adding all the edges in the neighbourhood of the vertices of the added edge. Doing this produces a valid peripheral edge ordering . Let us remind you that a peripheral edge is an edge such that it is not the middle edge of a chordless P4.

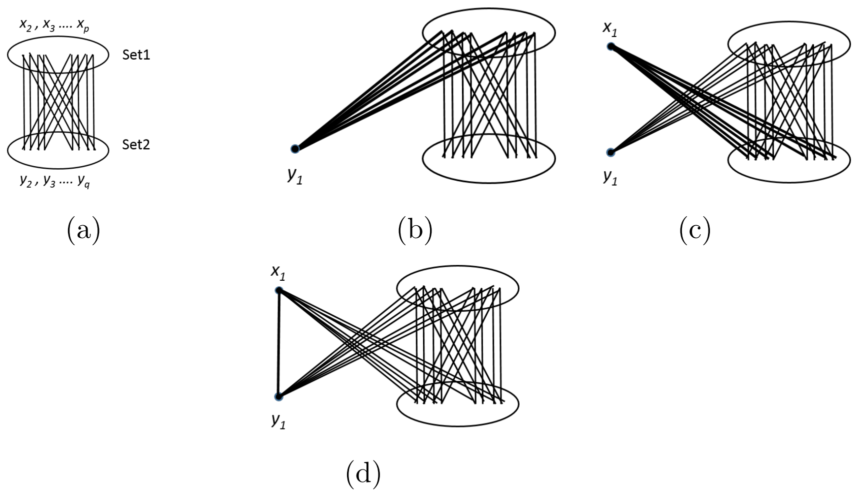


Figure 3.2: Perfect edge elimination ordering from 3.2a to 3.2d.

Proof of Correctness

The proof of the correctness of the algorithm can be done through Mathematical Induction. The proof assumes the correctness upto adding of the k-1 th edge in the reverse PEWVE Ordering and extends that to the k th edge. It is as follows:-

Base Case-

In case of k=1 as shown in the figure it is a peripheral edge as it is the lone edge and hence cannot be a middle edge of a chordless P4..

General Case-

Assume without loss of generality that the algorithm is correct upto the addition of the (k-1)th edge in the reverse of the PEWVE Ordering. Then if we add the edges incident on the vertices of this edge these edges are peripheral as these are the dangling edges that cannot be a middle edge. Also note that the edge has not been added thus far hence removing a possibility of the edges to be a middle edge in the graph constructed so far.

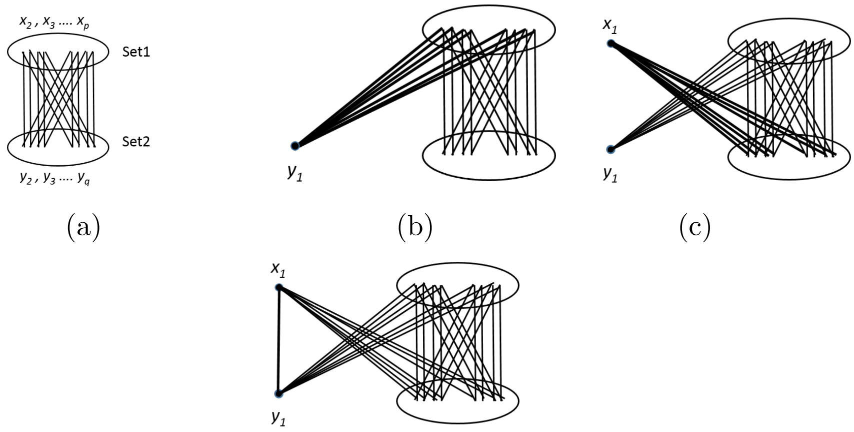
Now add the k th edge in the reverse of the PEWVE Ordering to the graph thus far. This is also a peripheral edge since by the definition of the bisimplical edge the neighbourhood of the vertices induce a complete bipartite graph which will serve as a chord for any potential P4 for which this edge is the middle edge. This has been explained further in the figure 3.2.

3.3.2 Complexity Analysis

It is clear to see that finding Peripheral Edge Ordering from PEWVE Ordering takes time linear in terms of no. of edges in the PEWVE Ordering. There can be as many as m edges in the PEWVE Ordering. To support this, consider a trivial case of a graph containing only two vertices joined by an edge. Hence we can find out Peripheral Edge Ordering from PEWVE Ordering for a Chordal Bipartite Graph in $O(m)$ time.
Original algorithm to find Peripheral Edge Ordering for a weakly triangulated graph took $O(n^2m)$ time. We can do this efficiently for Chordal Bipartite graphs in $O(min(m \log n, n^2))$ time by first finding PEWVE Ordering using Klask and Kratsch's algorithm and then applying our above described algorithm to find the Peripheral Edge Ordering.

3.4 Constructing Rigidity Tree Efficiently

Asish Mukhopadhyay *et al* [1] constructed Rigidity Tree based on reverse peripheral edge order of the given weakly triangulated graph. We build the graph starting from an empty graph by adding peripheral edges one by one in the reverse order of Peripheral Edge Ordering of the original weakly triangulated graph and simultaneously update the Rigidity Tree. Rigidity Tree construction includes addition of new nodes like triangles, quadrilaterals and $K_{2,3}$. Consider the peripheral edge e_i that is being newly added is between



(a) (b) (c) (d)

Figure 3.3: Perfect edge elimination ordering from 3.2a to 3.2d.

vertices x and y. Construction of rigidity tree has a bottleneck of finding no. of 3 length and 2 length paths between x and y which takes $O(n^2)$ time to find out the formation of new nodes and transformation of non-rigid components into rigid ones.

In our case of chordal bipartite graphs being bipartite, we do not need to consider formation of the triangles and there won't be any 2 length paths between x and y since bipartite graphs do not contain odd cycles. So, our only concern lies in finding out no. of 3 length paths between x and y more efficiently.

Depending on the no. of 3 length paths existing between x and y there can be different changes in the rigidity tree.

Case 1: *No. of 3 length paths = 0*

This is trivial case. The newly added peripheral edge e_i will be a dangling edge and hence there won't be any effect on the rigidity tree.

Case 2: *No. of 3 length paths = 1*

In this case a new quadrilateral is formed.

Chapter 4

Future Work and Conclusion

4.1 Conclusion

In the problem of generating weakly triangulated graph randomly, we have seen how to generate random rigidity tree. Also, we have brought together few techniques which can be used to solve various other problems of generating constrained weakly triangulated graphs.

For the second problem, the bottle neck of the original algorithm used for weakly triangulated graphs was calculating the peripheral edge order.We were able to reduce the time complexity of calculating peripheral edge order from $O(n^2m)$ to $O(min(mlogn, n^2))$

4.2 Future Work

Further towards solving generation of random weakly triangulated graph problem, we would like to generate the random maximally rigid sub-graph components which are currently only labelled such in the rigidity tree.

Future work for the second problem includes extending the idea and identifying the rigid and non-rigid components in a chordal bipartite graph and reducing the conplexity even further.We also want to extend the idea to another kind of graphs known as Laman Graphs as we have noticed some similarity between initial order Laman graphs and weakly triangulated graphs.

Bibliography

[1] S. V. Rao Ashish Mukhopadhyay. Linear layouts of weakly triangulated graphs. 2014.

[2] Hitoshi IBA. Random tree generation for genetic programming. *International Conference on Evolutionary Computation*, 2005.

[3] Liviu Panait Sean Luke. A survey and comparison of tree generation algorithms. 2001.

[4] D.Kratsch T.Kloks. Computing a perfect edge without vertex elimination ordering of a chordal bipartite graph. 1995.