Linear layouts of weakly triangulated graphs

Asish Mukhopadhyay[a], S.V. Rao[b,*], Sidharth Pardeshi[b], Srinivas Gundlapalli[b]

[a]*School of Computer Science, University of Windsor, Windsor, Ontario N9B 3P4, Canada*
[b]*Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, 781 039, Assam, India*

Abstract

A graph $G = (V, E)$ is said to be triangulated if it has no chordless cycles of length 4 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. Damaschke [1] showed how to compute all linear layouts of a triangulated graph, for a valid assignment of lengths to the edges of G. In this paper, we extend this result to weakly triangulated graphs, resolving an open problem. A weakly triangulated graph can be constructively characterized by a peripheral ordering of its edges. The main contribution of this paper is to exploit such an edge order to identify the rigid and non-rigid components of G. We first show that a weakly triangulated graph without articulation points has at most 2^{n_q} different linear layouts, where n_q is the number of quadrilaterals (4-cycles) in G. When G has articulation points, the number of linear layouts is at most $2^{n_b-1+n_q}$, where n_b is the number of nodes in the block tree of G and n_q is the total number of quadrilaterals over all the blocks. Finally, we propose an algorithm for computing a peripheral edge order of G by exploiting an interesting connection between this problem and the problem of identifying a two-pair in $\overline{G}$. Using an $O(nm)$ time solution for the latter problem when G has n vertices and m edges, we propose an $O(n^2m)$ time algorithm for computing its peripheral edge order. For sparse graphs, the time-complexity can be improved to $O(m^2)$, using the concept of handles [2].

Keywords: Rigidity, Graph embedding, Weakly triangulated graphs

*Corresponding author

Email addresses: `asish.mukerji@gmail.com` (Asish Mukhopadhyay), `svrao@iitg.ernet.in` (S.V. Rao), `s.pardeshi@alumni.iitg.ernet.in` (Sidharth Pardeshi), `gundlapalli@iitg.ernet.in` (Srinivas Gundlapalli)

1. Introduction

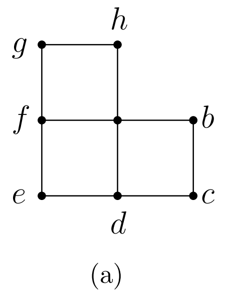
The problem we study in this paper is a restricted version of the point placement problem for which we seek to determine the locations of a set of distinct points on a line, uniquely up to translation and reflection, by making the fewest pairwise distance queries [3]. In the linear layout problem, we are given a set of pairwise distances (in the form of a weighted graph[1]) and the problem is to determine all possible placements of the vertices of the graph on a line or their linear layouts. The linear layout problem, in turn, is a special case of the graph embedding problem. Given a weighted, undirected graph $G = (V, E)$ and a positive integer k, an embedding of G in a k-dimensional Euclidean space, E^k, is a mapping, f, of V into E^k such that the weight of an edge $e = \{u, v\}$ in G is equal to the Euclidean distance between $f(u)$ and $f(v)$. For $k = 1$, such an embedding corresponds to a linear layout. Saxe [4] showed that the problem of embedding a weighted (incomplete) graph $G = (V, E)$ in Euclidean k-space is strongly NP-complete. Indeed, it remains so even when $k = 1$ and the edge weights are restricted to the values $\{1, 2\}$. Barvinok [5] showed that if G is k-embeddable for some k then it is k-embeddable for $k = \lfloor \sqrt{(8|E| + 1)} - 1)/2 \rfloor$, while Alfakih and Wolkowicz [6] gave an algorithm for constructing such a k-embedding. Hastad et al. [7] studied the approximability of the matrix-to-line problem, which is in essence the 1-embeddability problem.

Despite the strongly negative result of Saxe, it is possible to solve this problem in polynomial time for special classes of graphs with well-defined assignment of weights to their edges as we show below.

A graph $G = (V, E)$ is said to be triangulated (also called chordal) if it has no chordless cycle of length four or more. Damaschke [1] described an algorithm for generating all linear layouts of a triangulated graph given a valid assignment of lengths, l, to the edges of the graph. 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. We indicate this by writing (G, l). Note that G must be a simple graph for a valid assignment of lengths l. It was left as an open problem to extend this algorithm to other classes of graphs with weaker chordality properties.

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. A hole in G is an induced cycle on five or more vertices and an anti-hole is the complement of a hole. Alternatively,

[1]In this paper the terms length and weight, as applied to an edge of a graph, are used interchangeably, reflecting the practice in the surveyed literature.



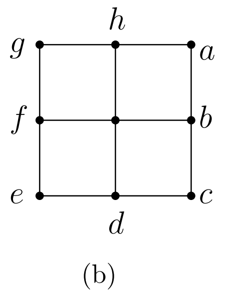


Figure 1: Some example graphs: (a) A weakly triangulated graph and (b) A graph that is not weakly triangulated.

G is weakly triangulated if it does not contain a hole or an anti-hole. Fig. 1(a) shows a weakly triangulated graph; however, the graph in Fig. 1(b) is not a weakly triangulated one as the outer boundary is a chordless 8-cycle. The class of weakly triangulated graphs includes the class of triangulated graphs as well as their complements.

In this paper, we show how to enumerate linear layouts for weakly triangulated (or weakly chordal) graphs. For the rest of this paper we will assume that G is a graph with m edges and n vertices.

2. Preliminaries

An edge of a graph is *peripheral* if it is not the middle edge of any P_4 (a 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 1. [8] *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 [8]:

Theorem 2. *A graph is weakly triangulated if and only if it admits a peripheral edge order.*

The graph of Fig. 1(b), for example, does not admit a peripheral edge order. Assume otherwise. If the last edge added in the peripheral edge order is not a boundary edge, the graph prior to the addition of this edge is not weakly triangulated. If the last edge added is a boundary edge, then it is clearly the middle edge of a P_4.

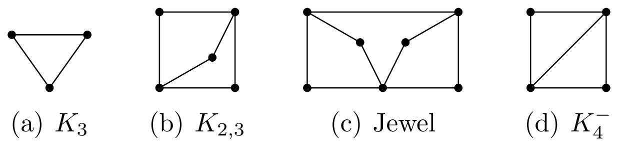


Figure 2: Some examples of line rigid graphs.

Let l be a valid assignment of lengths to the edges of G. By definition, (G, l) is said to be line rigid if there is a placement of the vertices G on a line that is unique up to translation and reflection. G is said to be line rigid if (G, l) is line rigid for every valid l. Some examples of line rigid graphs are shown in Fig. 2. Except for what is known as the jewel graph (Fig. 2(c)), the rest are also weakly triangulated. A quadrilateral is not line rigid as a length assignment l that makes it a parallelogram has two different layouts (Fig. 3).

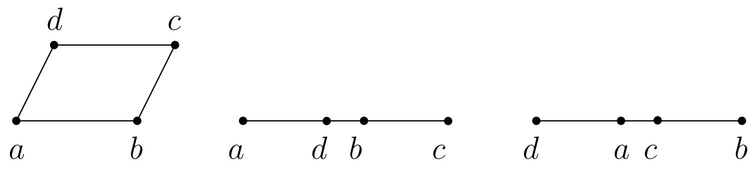


Figure 3: Two different placements of a parallelogram $abcd$

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* [9].

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 edge $\{u, v\}$ in Fig. 4(a) is a hinge edge. Hinge components hanging from the hinge edge $\{u, v\}$ are shown in Fig. 4(c).

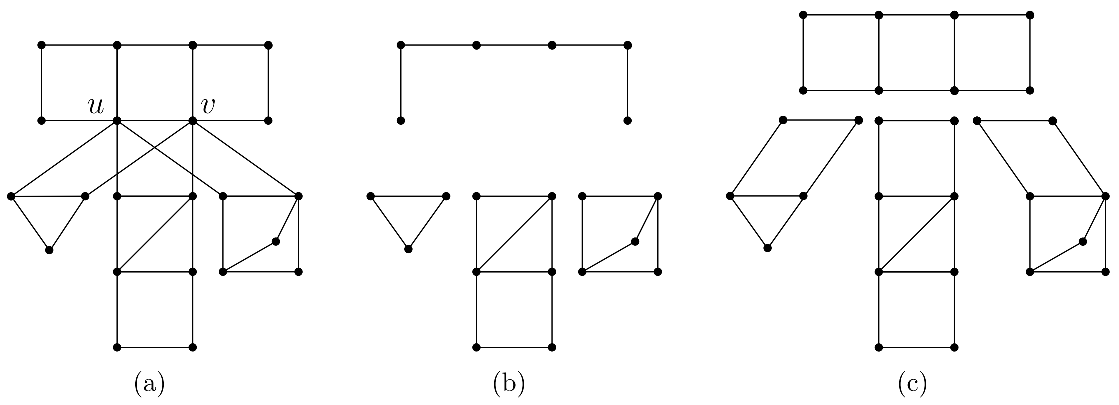


Figure 4: Hinge edge $\{u, v\}$: (a) Weakly triangulated graph, (b) Graph after deleting u and v, and (c) Hinge Components.

We assume that G is weakly triangulated graph without articulation points or hinge edges and has a valid assignment of edge lengths. Also, we freely use geometric terminology like line segments, triangles and quadrilaterals in lieu of edges, 3-cycles and 4-cycles.

3. Rigidity structure

Given a peripheral edge order of G, we reconstruct G by adding back the edges in reverse peripheral order. During this reconstruction, we identity the formation of new rigid and non-rigid components, and the conversion of non-rigid components into rigid ones.

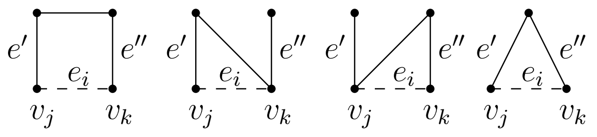


Figure 5: Connectivity of edges e' and e''.

Let G_i be the graph obtained by adding to G_{i-1} the i-th peripheral edge $e_i = \{v_j, v_k\}$. The edge e_i does not change the rigidity structure of the graph G_{i-1} if it is added to a maximally rigid subgraph of G_{i-1}. Otherwise, it changes the rigidity structure of G_{i-1} by way of forming a new rigid or non-rigid component or by changing a non-rigid component to a rigid one.

If only v_j is incident to G_{i-1}, then e_i is a dangling edge. Otherwise, e_i joins two vertices of G_{i-1}, creating some new cycles. To determine these, consider the set of edges E_j and E_k incident respectively on v_j and v_k. If e' is an edge in E_j and e'' an edge in E_k then as e_i is a peripheral edge, either the edges e' and e'' share a common end point or there is a chord between their end-points as in Fig. 5.

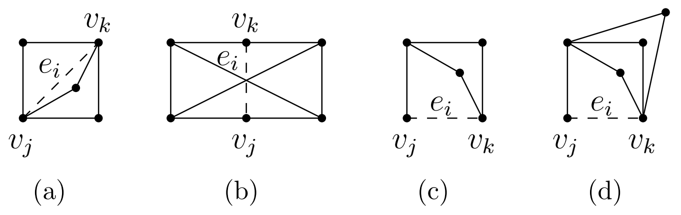


Figure 6: Different length paths between v_j and v_k: (a) Triples of paths of length two and (b) pair of paths of length three make the configurations rigid before adding e_i, (c) Pair of paths of length three with a common edge forms $K_{2,3}$ with e_i, and (d) Three paths of length three with a common edge forms $K_{2,4}$ with e_i.

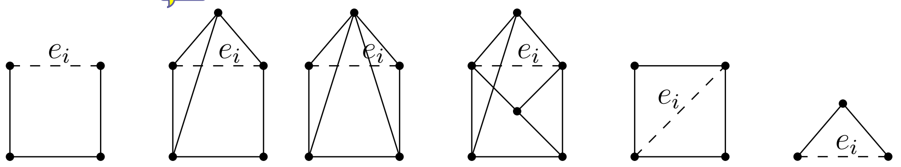
Thus the addition of e_i generates new triangles and quadrilaterals. If there are more than two paths of length two between v_j and v_k, excluding the chord e_i, each set of three paths group to form a rigid subgraph, $K_{2,3}$ (see Fig. 6(a)). Edge e_i gets added to a maximally rigid subgraph and no new rigid or non-rigid components are formed. Otherwise, new rigid or non-rigid components may be formed with at most two paths of length two between v_j and v_k.

Let us examine the collection of quadrilaterals generated, each quadrilateral corresponding to a path of length three between v_j and v_k, bounded by e_i. In order that pairs of such paths, with no common edge, keep the graph weakly triangulated, there must be chords between two pair of vertices as shown in Fig. 6(b). These give rise to two rigid subgraphs $K_{2,3}$, which makes the configuration rigid before adding e_i. Fig. 6(c) shows that if these two paths share an edge, then the addition of e_i creates a $K_{2,3}$. In other words, a non-rigid quadrilateral and a dangling edge form a $K_{2,3}$ with e_i. More than j paths of length three with a common shared edge, as in Fig. 6(d), implies a rigid $K_{2,j}$ for $j > 2$ with a dangling edge. Addition of e_i forms $K_{2,j+1}$, which makes the dangling edge rigid.

The rest of the section focuses on rigid and non-rigid components formed when there are at most one path of length three and two paths of length two between v_j and v_k. There are five different cases as shown in Fig. 7.

Case 1 (zero path of length two and one path of length three): In this case, in conjunction with e_i, we generate a quadrilateral.

Case 2 (one path of length two and one path of length three): In this case to maintain the weak triangulation property, there must be one or two chords



(a) Case 1 (b) Case 2 (c) Case 3 (d) Case 4 (e) Case 5

Figure 7: All combinations of paths of lengths two and three.

between pairs of vertices as shown in Fig. 7(b). The addition of e_i to the first configuration gives rise to a rigid graph - a $K_{2,3}$ with an additional edge; whereas the second configuration is rigid before addition of e_i.

Case 3 (two paths of length two and one path of length three): In this case, each path of length two together with the path of length three creates a 5-cycle which must be chorded to preserve the weak triangulation property. Since one or two chords can be added in each 5-cycle between the vertices of these paths, as shown in the previous case, nine cases arise (see Fig. 8); of these, two have the configuration shown in Fig. 7(c); the configuration of the remaining seven cases (marked by $\times$) are rigid prior to the addition of e_i. The former two configurations cannot arise as the complement graph has a chordless 6-cycle. Thus in this case e_i is added to a rigid component.

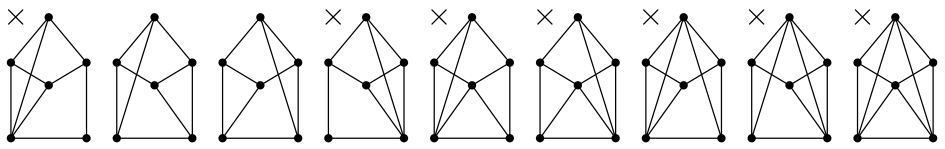


Figure 8: Nine different ways of chording two 5-cycles.

Case 4 (two paths of length two and zero path of length three): In this case the addition of edge e_i triangulates a quadrilateral, generating a rigid graph.

Case 5 (one path of length two and zero path of length three): In conjunction with e_i we have a triangle, a rigid graph.

Thus in the first case we generate a single non-rigid component, viz., a quadrilateral; in the remaining cases we generate rigid components: triangles, triangulated quadrilaterals, $K_{2,3}$'s. Moreover, depending on the number of different cycles between v_j and v_k, we can identify the various changes in the rigidity structure during the reconstruction of G. These are summarized in Table 1.

Number of paths of lengths three and two	Figure	Change of rigidity structure due to the addition of e_i
0, 0		e_i is a dangling edge.
0, 1	Fig. 7(e)	a new rigid component, viz., a triangle, is formed.
0, 2	Fig. 7(d)	a non-rigid quadrilateral changes into the rigid graph K_4^-.
0, ≥ 3	Fig. 6(a)	e_i is added to a rigid component.
1, 0	Fig. 7(a)	a non-rigid quadrilateral is formed.
1, 1	Fig. 7(b)	1st **configuration**: quadrilateral and triangle forms a rigid graph $K_{2,3}$ with an additional edge. 2nd **configuration**: e_i is added to a rigid component.
1, ≥ 2	Fig. 7(c)	e_i is added to a rigid component.
≥ 2, ≥ 0 no common edge among the paths of length three	Fig. 6(b)	e_i is added to a rigid component.
2, 0 with a common edge among the paths of length three	Fig. 6(c)	quadrilateral and dangling edge forms $K_{2,3}$.
≥ 2, ≥ 1 with a common edge among	Combinations of Fig. 6(c) or Fig. 6(d) with Fig. 7(b) or Fig. 7(c)	e_i is added to a rigid component.
≥ 3, 0 with a common edge among the paths of length three	Fig. 6(d)	dangling edge becomes a part of rigid component.

Table 1: Summary of changes in the rigidity structure.

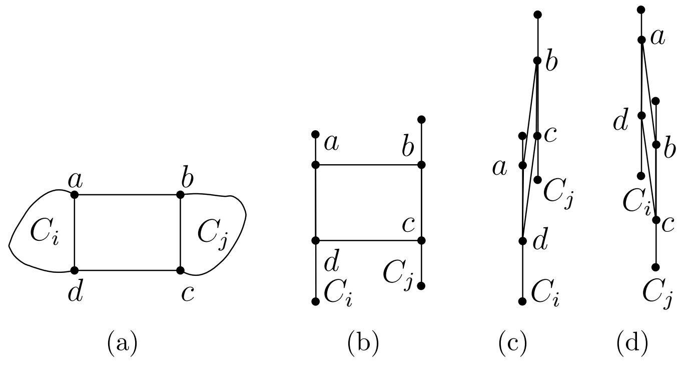


Figure 9: Non-rigid quadrilateral remains non-rigid: (a) Rigid components C_i and C_j are adjacent to a non-rigid quadrilateral, (b) Unique layouts of C_i and C_j are shown as vertical segments, and (c) and (d) are two different layouts of the quadrilateral *abcd*.

Note that a non-rigid quadrilateral remains non-rigid even if all its vertices are parts of maximally rigid components as shown in Fig. 9. This is true even if there are four different rigid components adjacent to a non-rigid quadrilateral, one on each side.

4. Rigidity tree

From the discussion of the previous section, it is clear that the only minimally line rigid subgraphs of a weakly triangulated graphs are the graphs $K_{2,3}$ and K_3 and the only non-rigid subgraph is a quadrilateral. We use these facts to determine the relationship between maximal rigid subgraphs and minimal non-rigid subgraph of a weakly triangulated graph. The interesting question is whether these can be found as we reconstruct the weakly triangulated graph by adding back the edges in reverse peripheral edge order, starting from an empty graph G_0.

Let $C = \{C_1, C_2, \ldots, C_l\}$ be the set of components of G, where each C_i, for $1 \leq i \leq l$, is either a quadrilateral or a maximal line rigid subgraph of G. We freely use maximal rigid component in lieu of maximal line rigid subgraph. The rigidity graph $R_G = (V_G, E_G)$ is a graph whose nodes are the components of G and there is an edge connecting two nodes if the corresponding components share an edge. The set of rigid components and rigidity graph of the weakly triangulated graph of Fig. 10(a) are shown in Fig. 10(b) and Fig. 10(c) respectively. We show that R_G is a tree by exploiting the interaction between non-rigid components and maximally rigid components.

Lemma 1. *Two quadrilaterals can share at most one edge.*

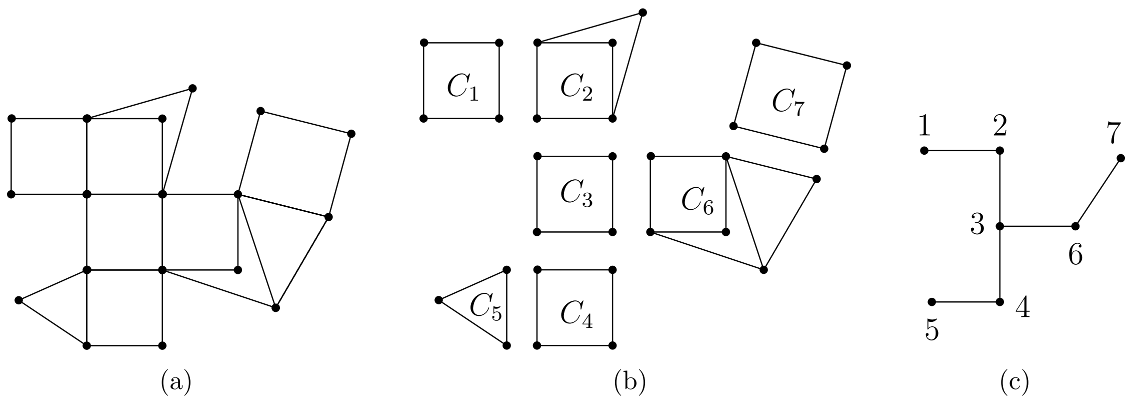


Figure 10: (a) Weakly triangulated graph G, (b) Components of G, and (c) Rigidity tree of G.

Proof: If two distinct quadrilaterals C_i and C_j share two edges then they form a $K_{2,3}$. This implies that $C_i \cup C_j$ must be a subgraph of a maximally rigid component. This contradicts the fact that C_i and C_j are distinct components. If three edges are common then the distinct edges of C_i and C_j are parallel, which is also a contradiction since G is assumed to be a simple graph. This proves the lemma. ▮

Lemma 2. *No edge is common to two maximal line rigid subgraphs.*

Proof: There cannot be an edge common to two maximally line rigid subgraphs because both have a linear layout on a supporting line of the common edge, fixing the placements of all the vertices. This contradicts the definition of the maximality of the rigid components. ▮

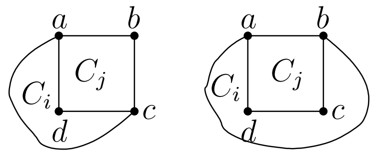


Figure 11: Interaction between a maximally rigid component C_i and a non-rigid component C_j

Lemma 3. *A quadrilateral and a maximal line rigid subgraph can share at most an edge.*

Proof: If more than one edge is common between a maximal line rigid subgraph C_i and a quadrilateral C_j (see Fig. 11), then C_i and C_j must be part of the same maximal line rigid subgraph. This contradicts the definition of maximality of the line rigid subgraph C_i. ▮

One way of formation of a cycle in R_G is by cycle of rigidity components. If an edge e_i is common to $k \geq 3$ components, it forms a k-clique in R_G, which contains p-cycle for $3 \leq p \leq k$. Respectively call these cycles as component cycle and clique cycle. We show below, R_G does not contain any cycle.

Lemma 4. *There is no component cycle of length two in R_G.*

Proof: A component cycle of length two in R_G between C_i and C_j implies that C_i and C_j have two common edges, which is not possible because of the lemmas 1, 2, and 3. ▮

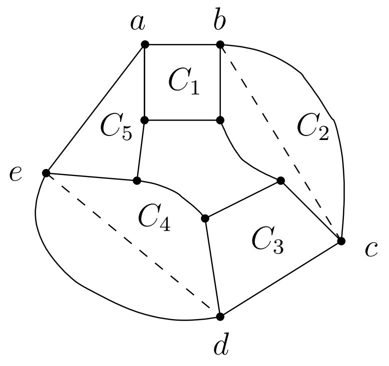


Figure 12: Five components forming cycle.

Indeed, we show that there is no component cycle of any length in R_G.

Lemma 5. *The rigidity graph R_G of a weakly triangulated graph without articulation points and hinge edges does not contain component cycles.*

Proof: Assume there is a 5-cycle in R_G. It contains at most two maximal line rigid subgraphs and at least three non-rigid components, since there is no common edge between any two maximal line rigid components. A 5-cycle formed by two maximal line rigid subgraphs and three quadrilaterals is shown in Fig. 12. This contradicts the definition of a weakly triangulated graph, since there is a cycle $\{a,b\}, \widehat{\{b,c\}}, \{c,d\}, \widehat{\{d,e\}}, \{e,a\}$ with at least five edges, where the notation $\widehat{\{b,c\}}$ denotes a chordless path between b and c. This argument extends to all cycles of length ≥ 5 formed with different types of components.

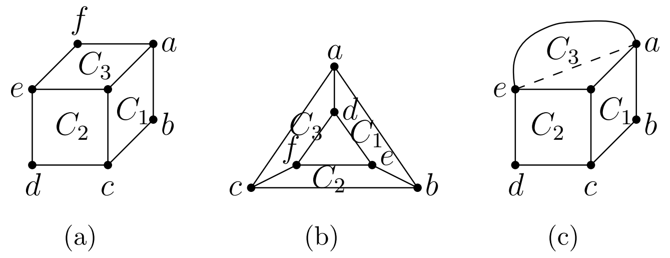


Figure 13: Three components cycles: (a) Three non-rigid components cycle with a 6-cycle, (b) Three non-rigid components cycle whose complement is a 6-cycle, and (c) Two non-rigid and a maximally rigid component with a minimum cycle length of five.

The nonexistence of 3-cycles and 4-cycles in R_G can be proved by enumerating all possible cycles with different types of components.

All three different types of 3-cycles possible in R_G are shown in Fig. 13. All these 3-cycles violate the definition of a weakly triangulated graph. The case of Fig. 13(a) cannot happen because it contains a chordless 6-cycle $\{a,b\},\{b,c\},\{c,d\},\{d,e\}$, $\{e,f\}$, $\{f,a\}$ in G. The case of Fig. 13(b) cannot happen because its complement $\{a,e\},\{e,c\},\{c,d\}$, $\{d,b\}$, $\{b,f\}$, $\{f,a\}$ is a 6-cycle in $\overline{G}$. Finally, the case of Fig. 13(c) cannot happen because it contains a chordless cycle $\{a,b\}$, $\{b,c\}$, $\{c,d\}$, $\{d,e\}$, $\widehat{\{e,a\}}$ in G, of length at least five.

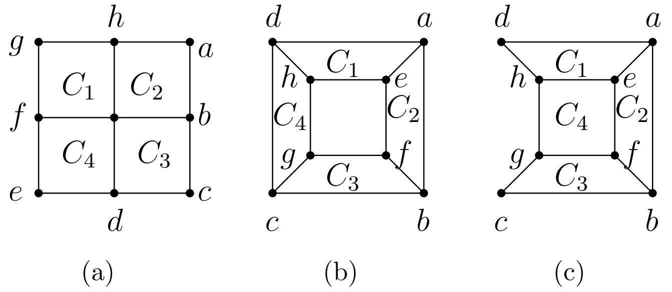


Figure 14: Four non-rigid components cycles: (a) Contains an 8-cycle, and (b) and (c) contains a 6-cycle.

All three different types of 4-cycles possible with non-rigid components are shown in Fig. 14. The graph of Fig. 14(a) contains a chordless 8-cycle $\{a,b\},\{b,c\},\{c,d\}$, $\{d,e\}$, $\{e,f\},\{f,g\},\{g,h\},\{h,a\}$ in G, and the graphs of Figs. 14(b) and 14(c) contain a chordless 6-cycle $\{a,b\},\{b,f\},\{f,g\},\{g,h\}$, $\{h,d\}$, $\{d,a\}$. The proofs for 4-cycles in R_G formed with non-rigid and maximal line rigid subgraphs are similar.

▮

Lemma 6. *No edge is common to more than two components.*

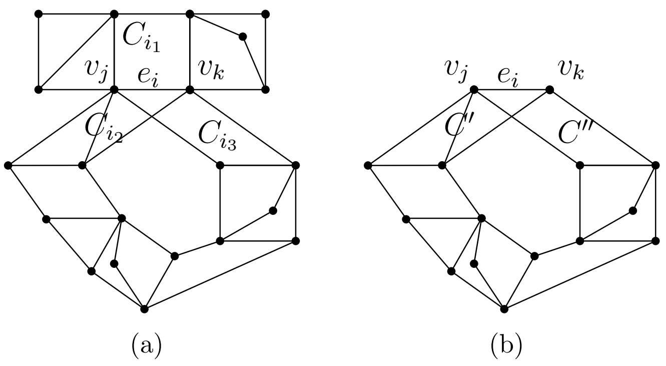


Figure 15: Clique cycle: (a) Edge e_i is common to three components, which forms 3-clique in rigidity graph, and (b) Its subgraph G''' with two components, which forms a component cycle in rigidity graph.

Proof: Assume otherwise. There exists an edge $e_i = \{v_j, v_k\}$, which is common to $k \geq 3$ rigidity components $C_{i_1}, C_{i_2}, \ldots, C_{i_k}$. The graph G is disconnected into at most two components by removing the vertices v_j and v_k and all the edges incident on them, because e_i is not an hinge edge. Assume G is disconnected into two components G' and G''. At least two rigidity components in $\{C_{i_1}, C_{i_2}, \ldots, C_{i_k}\}$ belongs to either G' or G''. Without loss of generality, let C' and C'' be these two rigidity components and belongs to G'.

Let G''' be the graph resulted by adding e_i and the edges incident on v_j and v_k from the vertices of G' to G'. G''' contains a cycle of rigidity components as shown in 15(b), which contradicts Lemma 5.

We get similar contradiction if G is not disconnected. Hence follows the lemma.

▮

The lemmas 4, 5, and 6 imply the following theorem.

Theorem 3. *The rigidity graph of a weakly triangulated graph without articulation points and hinge edges is a tree.*

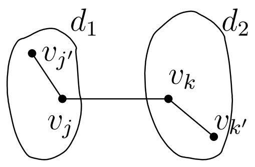


Figure 16: R_{G_i} is connected.

5. Rigidity tree construction

To enumerate all linear layouts of a weakly triangulated graph G, we have to construct its rigidity tree R_G, discussed in the previous section. In this section, we propose an efficient algorithm for this.

The main idea underlying the algorithm is to construct this tree incrementally in parallel with the reconstruction of G by adding edges in reverse edge peripheral order. As we saw in Section 3, the addition of an edge changes the rigidity structure of the partially constructed graph. To reiterate, this can generate a new quadrilateral, change a non-rigid component into a rigid one, is added to a maximally rigid component or is just a dangling edge. Each of these possibilities is termed an event when we update the rigidity tree with the help of suitable data structures.

We maintain an adjacency matrix $A_{G_i}[n \times n]$ for the partially constructed graph G_i (G_0 is the empty graph and $G_m = G$). Besides adjacency information, we store in each entry the index of the peripheral edge $e_i = \{v_j, v_k\}$, and pointers p_1^{jk} and p_2^{jk} to the (at most) two rigidity components to which e_i belongs.

In each node of the partially constructed rigidity tree R_{G_i}, besides the label rigid or non-rigid and pointers to its tree neighbors, we also maintain a pointer p_3 to an appropriate representation of this rigidity component. A dynamic list of edges and sorted list of vertices that make up this component would be an appropriate representation; in this list, edges shared with tree neighbors are marked as common. Each rigid component also maintains the placement of its vertices. Each node in R_{G_i}, marked rigid, maintains the placements of its common edges.

Before we describe our algorithm that uses and justifies the complex data structures above, we briefly digress to prove the following lemma, characterizing the connectivity of the partially constructed rigidity tree, R_{G_i}.

Lemma 7. *R_{G_i} is connected.*

Proof: Assume otherwise. Since the final rigidity tree R_G is connected, there exists two disconnected components d_1 and d_2 just before they are connected by the addition of a particular peripheral edge $e_i = \{v_j, v_k\}$. Assume without loss of generality that v_j and v_k are respectively belongs to rigidity components $C_1 \in d_1$ and $C_2 \in d_2$. There exists two edges $\{v_{j'}, v_j\}$ and $\{v_{k'}, v_k\}$ respectively in the rigidity components C_1 and C_2, as shown in Fig. 16, such that e_i is the middle of P_4, which is a contradiction. This proves the lemma. ∎

For each peripheral edge $e_i = \{v_j, v_k\}$, we find, using the adjacency matrix $A_{G_{i-1}}[n \times n]$, the number of 2-paths and 3-paths between v_j and v_k in G_{i-1}. The time complexities of these steps are in $O(n)$ and $O(n^2)$ respectively. This helps us identify the exact event affecting the rigidity structure, and use this information to update the relevant data structures as below.

If the event is the formation of a triangle or a quadrilateral, we insert a new node in the rigidity tree and assign a label to it (see Fig. 17). In both the cases, create the rigidity component c_{new} and update the rigidity component pointer for each edge of c_{new} in $A_{G_i}[n \times n]$. If the event is the formation of a $K_{2,3}$ or a quadrilateral is split, relabel the node as rigid (see Fig. 18).

Whenever a rigid component is formed, merge it with its rigid neighbors, if any. All neighbors of merged nodes are now also the neighbors of the newly created rigid node. In Figs. 17(b), 18(b), 18(d), and 18(f) the rigid components formed are, in each case, the neighbor of another rigid component. This merger is effected as follows. Find the linear layout of the newly formed rigid component, say c_{new}, in $O(1)$ time, since the number of edges in c_{new} is constant. An edge $e \in c_{new}$ can have two placements, if e also belongs to another rigid component c'. Determine the transformation of the placement of e in c' with respect to the placement of e in c_{new}. Apply the transformation to the terminal vertices of all the edges in c'. For each edge $e \in c'$, change the component pointer in $A_{G_i}[n \times n]$ to point to c_{new} instead of c'. This can be done in time proportional to number of edges in c'. Repeat these steps for each edge in c_{new}. The time complexity of this step is in $O(m)$, since the number of edges in c_{new} is a constant. The total time required for all m peripheral edges is thus in $O(m^2)$.

For each $e \in c_{new}$ add an edge between c_{new} and c'' in R_{G_i}, if $e \in c''$. The above discussion is summarized in the following result as a peripheral edge order can be computed in $O(n^2 m)$ time (see Section 7).

Theorem 4. *The rigidity tree R_G of a weakly triangulated graph without any articulation points and hinge edges can be constructed in $O(n^2 m)$ time.*

The number of linear layouts of a weakly triangulated graph without articulation points and hinge edges is 2^{n_q}, since each non-rigid quadrilateral has at most two distinct layouts, where n_q is number of non-rigid quadrilaterals in the rigidity tree.

Moreover, all the linear layouts of the graph can be generated using depth-first or breadth-first traversal of the rigidity tree.

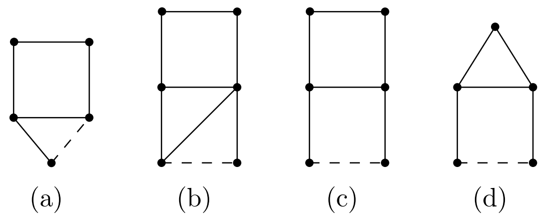


Figure 17: Two different ways of triangle and non-rigid quadrilateral formations are depicted in (a) and (b), and (c) and (d) respectively.

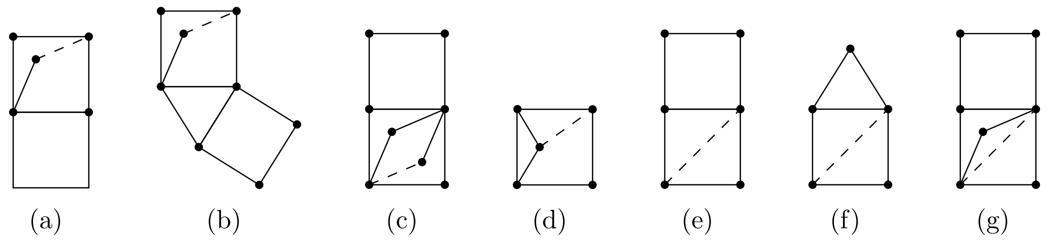


Figure 18: Four different ways of $K_{2,3}$ formation and three different ways of quadrilateral split are depicted in (a) to (d) and (e) to (g) respectively.

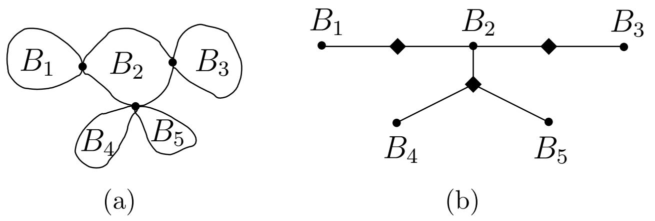


Figure 19: (a) Weakly triangulated graph with articulation points, and (b) Its block tree.

6. Arbitrary weakly triangulated graph

When G has articulation points, consider the block tree, T, of the graph G. If G has n_b blocks then the number of different layouts of the blocks is 2^{n_b-1} as each articulation point permits two different placements of a block in a linear layout.

Fig. 19(a) shows a graph G with three articulation points and five blocks; Fig. 19(b) shows the corresponding block tree. Fix a placement of the block B_2, then relative to it the remaining four blocks each has two different placements and thus 16 different placements.

Divide each block B_i into hinge components if it contains hinge edges as shown in Fig. 4(c). For each hinge components and blocks without hinge edge, construct the rigidity tree from their respective peripheral edge order.

We have already shown that the presence of q quadrilaterals within each block allow for 2^q different layouts. Hence we have at most $2^{n_b-1+n_q}$ linear layouts where n_q is the total number of quadrilaterals over all the blocks.

The time required to construct a block tree and divide each block into hinge components are in $O(n)$ and $O(m^2)$ respectively. Hence, we can construct the hierarchical structure of block tree, hinge components, and rigidity tree of an arbitrary weakly triangulated graph in $O(n^2m)$ time. All the linear layouts can be generated using this hierarchical structure.

7. Computing a peripheral edge order

Vertices $\{u, v\}$ of G are said to be a two-pair, if all chordless paths between u and v are of length two. It was shown in [8] that $\{u, v\}$ is a two-pair in the complement graph $\overline{G}$ if $\{u, v\}$ is a peripheral edge in G; furthermore, $(e_1, e_2, e_3 \ldots, e_m)$ is a peripheral edge order of G if and only if $(e_1, e_2, e_3 \ldots, e_m)$ is a two-pair non-edge order (end points of e_i are a two-pair) of $\overline{G}$. By Theorem 2 stated earlier, we also know that such a peripheral edge order exists for a weakly triangulated graph. The algorithms that we propose will generate the peripheral edge order in reverse; that is, if the output of the algorithm is $(e_m, e_{m-1}, \ldots e_1)$, then the peripheral edge order is $(e_1, e_2, \ldots e_m)$.

A brute-force scheme to obtain the reverse peripheral edge order of G goes thus. We first obtain a two-pair in the complement graph $\overline{G}$ and output the edge joining the corresponding vertices in G as the first peripheral edge. Next, remove this edge from G and insert it into $\overline{G}$. Repeat these two steps for the updated graphs until $\overline{G}$ is a complete graph, or correspondingly G becomes empty. Then the peripheral edge order is the reverse of the output order of the edges.

To compute a two-pair, we use the $O(nm)$ algorithm proposed in [10]. As we shall be referring to some of its details in the subsequent paragraphs, we digress to give a brief description below.

Let $N(v)$ be the neighborhood of a vertex v of G. Let $H_1, H_2, \ldots, H_k$ be a decomposition of $H = G - \{v\} - N(v)$ into connected components. The label of

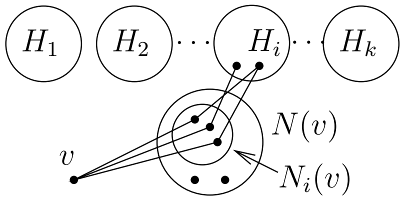


Figure 20: The Arikati-Rangan algorithm in a nutshell.

a vertex $u \in H$, $label(u)$, is i if it belongs to H_i. Let $N_1(v), N_2(v), \ldots, N_k(v)$ be a decomposition of $N(v)$ such that $N_i(v) = \{u | u \in N(v)$ and u is adjacent to some vertex in $H_i\}$ (see Fig. 20).

For $u \in H$, define $NV(u) = \{w | w$ is adjacent to u and in $N(v)\}$. Then $\{u, v\}$ is a two-pair if $NV(u) = N_{label(u)}(v)$.

To continue, since the input graph is $\overline{G}$, the time-complexity for a two-pair computation is in $O(n^3)$. As G reduces to an empty graph over m iterations, the time complexity of the brute-force algorithm is in $O(mn^3)$.

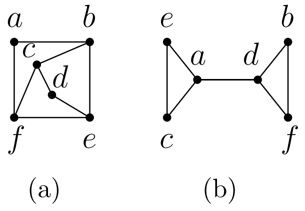


Figure 21: Iteration 1: (a) Graph G, (b) its complement.

The above brute-force algorithm makes a lot of redundant computations by running the Arikati-Rangan algorithm in each iteration. To pin down the redundancy precisely, consider a small graph and its complement (Fig. 21).

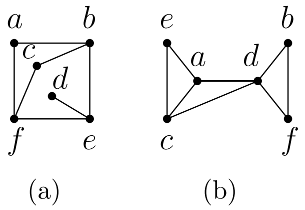


Figure 22: Iteration 2: (a) Graph G, (b) its complement.

After a first round of the Arikati-Rangan algorithm, if the two-pair returned is

$\{c, d\}$, then the updated graphs G and its complement for the next round are shown in Fig. 22. During the first-round (or the i-th round, in general) of computations, we need to store the intermediate results of the algorithm so that these can be used in the next round. For example, when we are considering the node c in the complement graph $\overline{G}$, we need to store the components H_k that are produced after removing nodes c and its neighbors (i.e. a and e) from the complement graph, so that these can be used in the next round. In this way, all the intermediate results are stored.

It is clear from Fig. 22 that the principal way in which the current round of the two-pair algorithm differs from the previous lies in how we deal with the nodes c and d and the ones adjacent to these. We elaborate on this below.

Consider the i-th round of our incremental version of the Arikati-Rangan algorithm. Assume that the edge $\{u, v\}$ was returned by the previous round. The two-pair algorithm considers each node, and computes the neighborhood-deleted components of that node. Depending on the node being processed in the present round, three cases arise:

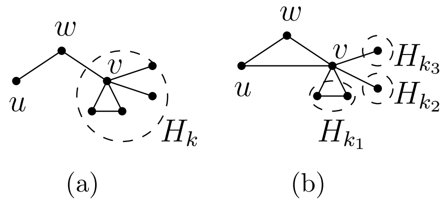


Figure 23: Components H_k of the graph $\overline{G}$: (a) iteration i and (b) iteration $i + 1$.

Case 1 (Node u or v is being processed): In this case, node u (v respectively) has a new neighbor because of the newly-added edge $\{u, v\}$. Also, the components H_k can change (see Fig. 23). For this case, we will just redo the two-pair algorithm for the respective nodes u and v.

Case 2 (Node u and v are not neighbors of the node s being processed): In this case, the new edge could not have modified the neighbor set of the node being considered. The only change it can effect is to join at most two components H_i and H_j. If a disjoint set data structure is used, then the union of two sets can be done in $O(n)$ time. We will also need to modify the set $N_i(v)$ for these two sets H_i and H_j, while the neighborhood of all other H_k's remain the same. This takes $O(n)$ time since the values of $NV(p)$ remain the same for all nodes p.

Case 3 (Node u or v is a neighbor of the node s being processed:) In this case, the new edge could not have modified the neighbor set of the node being considered. Neither could it modify the components H_k's because when we remove the node and its neighbors, we will get the exact same subgraphs as was considered in the previous iteration. The only change will be in the set $N_i(s)$. Therefore, if the node u is a neighbor of s, then the component H_i which contains the node v will now also have u in its set $N_i(v)$. Also, the set $NV(v)$ will now also include the node u. Both of these operations can be done in $O(n)$ time.

If both the nodes u and v are neighbors of s, it is easy to see that there will be no change in any of the components or sets.

We consolidate all of the above discussion and other details into **Algorithm 1**.

7.1. Time Complexity

Steps 2, 4, and 5 take $O(1)$ time if we are given the adjacency matrix. Step 1 takes $O(n^2)$ time to compute, and Step 3 takes $O(n^3)$ time to compute (remember that the complement graph can have as many as $O(n^2)$ edges).

Each of the if conditions in Step 6.2.2 and Step 6.2.3 for case 2 and case 3 takes at most $O(n)$ time. But the if condition for case 1 in Step 6.2.1 takes $O(n^2)$ time (note that we are only executing one iteration of the two-pair algorithm). Also the for loop in Step 6.2.4 is executed for each node, i.e. n times. The total time that Step 6.2 takes is $O(n^2)$ though since case 1 occurs for only two nodes (the ones in which the edge was added), and for the remaining $n-2$ nodes, it will be $O(n)$ time. Thus, in total we spend only $O(n^2)$ time for each iteration. Also, the *while* loop in Step 6 is executed m times, since after each iteration one edge is removed from the graph G. Therefore, the time complexity of our algorithm is in $O(n^2m)$.

Hayward et al. [2] proposed an algorithm that employs the interesting concept of handles for recognizing weakly triangulated that runs in $O(m^2)$ time. We can obtain a peripheral edge order in the same time. When G is sparse, we can use this result to obtain a more efficient algorithm.

8. Conclusions

In this paper we have brought together an interesting mix of techniques to solve the problem of generating all linear layouts of weakly triangulated graphs. One offshoot of this is the light that it throws on the rigidity structure of weakly triangulated graphs and whether this can be exploited in other algorithmic applications of this class of graphs. It would also be interesting to find some practical applications of this

```
Algorithm 1: Peripheral-Edge-Ordering
  Data: A graph $G = (V, E)$, and adjacency lists denoted by $Adj(v), v \in V$.
  Result: A peripheral edge order of the graph, if it exists.
  (1) Compute the adjacency matrix of the graph $G = Adj(G)$ and the complement
        graph $= Adj(\overline{G})$.
  (2) Initialize the list that will contain the peripheral edge order as $PE = \varnothing$
  (3) Run the two-pair algorithm on the complement graph $\overline{G}$, let it return the pair
        $\overline{uv}$. Also, while running the algorithm store all intermediate values. (in
        particular, for each node $v$, store the corresponding components $H_k$'s, $NV(u)$,
        $N_i(u)$ and $label(u)$ for all nodes $u \neq v$).
  (4) Add the edge $\{u, v\}$ to the list $PE$
  (5) Remove the edge $\{u, v\}$ from the graph $G$.
  (6) While $G$ contains some edge do
        (6.1) Update the adjacency matrix of the complement graph $\overline{G}$ denoted by
                $Adj(\overline{G})$
        (6.2) for $i = 1$ to $n$ do
                (6.2.1) if node $i$ belongs to Case 1 then
                            Run the two-pair algorithm for this node and
                            update $H_k$, $NV(u)$, $N_i(u)$ and $label(u)$.
                (6.2.2) if node $i$ belongs to Case 2 then
                            Get the two components in which $u$ and $v$ belong.
                            Obtain the set $N_i(v)$ which is the union of these two sets.
                            Also make a new set $H_k$ which is a union of these sets.
                (6.2.3) if node $i$ belongs to Case 3 then
                            Update the value of $N_i(v)$ and $NV(u)$ for that component
                            in which a new incident edge got added.
                (6.2.4) for all $u \in H$ do
                            if $|N_{label(u)}(v)| = |NV(u)|$ then
                               declare $(u, v)$ is a two-pair and STOP;
        (6.3) Append the edge $\{u, v\}$ to the list $PE$.
        (6.4) Remove the edge $\{u, v\}$ from the graph $G$
```

algorithm. Several open problems remain. One is to explore if the technique can be extended to other classes of graphs that have similar constructive characterizations. Another problem is to consider generating two-dimensional layouts of graphs.

9. Acknowledgements

This work is supported by an National Science and Engineering Research Council of Canada Discovery Grant to the first author. It was done during the first author's sabbatical visit from July, 2012 to September, 2012 to the Indian Institute of Technology, Guwahati, India. The support provided by the Indian Institute of Technology, Guwahati is gratefully acknowledged.

References

[1] P. Damaschke, Point placement on the line by distance data, Discrete Applied Mathematics 127 (1) (2003) 53–62.

[2] R. B. Hayward, J. P. Spinrad, R. Sritharan, Improved algorithms for weakly chordal graphs, ACM Trans. Algorithms 3 (2). `doi:10.1145/1240233.1240237`. URL `http://doi.acm.org/10.1145/1240233.1240237`

[3] M. S. Alam, A. Mukhopadhyay, Improved upper and lower bounds for the point placement problem, Tech. rep., University of Windsor (2010).

[4] J. B. Saxe, Embeddability of weighted graphs in k-space is strongly NP-hard, in: 17th Allerton Conference on Communication, Control and Computing, 1979, pp. 480–489.

[5] A. I. Barvinok, Problems of distance geometry and convex propeties of quadratic maps, Discrete and Computational Geometry 13 (1995) 189–202.

[6] A. Y. Alfakih, H. Wolkowicz, On the embeddability of weighted graphs in euclidean spaces, Tech. rep., University of Waterloo (1998).

[7] J. Hastad, L. Ivansson, J. Lagergren, Fitting points on the real line and its application to rh mapping, Journal of Algorithms 49 (2003) 42–62.

[8] R. Hayward, Generating weakly triangulated graphs, J. Graph Theory 21 (1996) 67–70.

[9] R. Diestel, Graph Theory, Springer, 2005.

[10] S. R. Arikati, C. P. Rangan, An efficient algorithm for finding a two-pair, and its applications, Discrete Applied Mathematics 31 (1) (1991) 71–74.