Computing Rigid Components of Pseudo-triangulation Mechanisms in Linear Time *

Jack Snoeyink† Ileana Streinu‡

Abstract

We investigate the problem of detecting rigid components (maximal Laman subgraphs) in a pseudo-triangulation mechanism and in arbitrary pointed planar frameworks. For general Laman graphs with some missing edges, it is known that rigid components can be computed in $O(n^2)$ time. Here we make substantial use of the special geometry of pointed pseudo-triangulation mechanisms to achieve linear time.

The main application is a more robust implementation and a substantial reduction in numerical computations for the solution to the Carpenter's Rule problem given by the second author.

1 Introduction

We present an efficient linear time algorithm for computing rigid components in pointed planar graphs, in particular in pointed pseudo-triangulation mechanisms.

Detecting rigid components in frameworks has emerged as an important subproblem in applications ranging from the study of flexibility in glass networks [8] and protein chains [7, 11], to collision-free motion planning of planar robot arms [12]. Since some of these applications deal with special classes of graphs, it is natural to investigate the complexity of the problem in these classes. Most notably, we are interested in the complexity of detecting rigid components in planar graphs (which remains an open problem) and in pseudo-triangulation mechanisms (solved in this paper). This latter case was posed as an open question [13], since its efficient solution reduces the amount of algebraic computation in the robot arm motion planning algorithm of [12].

A planar graph embedded with straight edges is *pointed* if each vertex is incident to some angle strictly greater than π — we call this the *large angle* incident to the vertex. A *pseudo-triangle* is a simple polygon in the plane that has exactly three convex internal angles (i.e. that measure less than π). A *pseudo-triangulation* of a set of points P covers the convex hull of P with pseudo-triangles having vertices from P and disjoint interiors. We will assume that P is in general position, with no three points collinear, so that angles of 0 or π never arise.

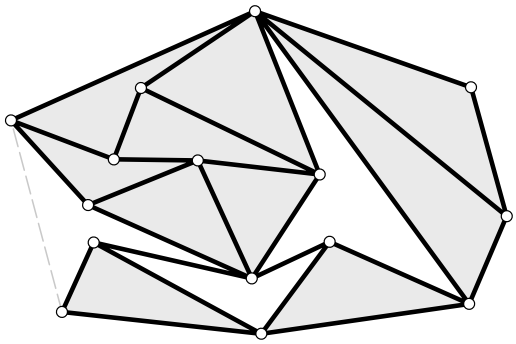

Figure 1: A pseudo-triangulation mechanism formed by deleting the dashed convex hull edge. The mechanism has 6 rigid components bounded by convex cycles: four shaded polygons and two single edges.

Pseudo-triangulations are rigid when viewed as frameworks with fixed length bars for edges and rotatable joints for vertices. The only continuous motions that preserve the lengths of all edges of the graphs are induced by the Euclidean motions of the vertex set P as a whole. Pointed pseudo-triangulations are minimally rigid—the deletion of any edge produces a flexible framework with one degree of freedom. A pointed pseudo-triangulation missing several edges will be called a *pointed mechanism*. Particularly relevant are those obtained by the deletion of exactly one convex hull edge, as in Fig. 1: these will be called *pseudo-triangulation mechanisms*. They have a one-degree-of-freedom *expansive motion* [12] which increases the length of the removed convex hull edge, preserves the lengths of all remaining edges, and never decreases the distance between any pair of vertices.

The *rigid components* of a flexible framework are the maximal sets of vertices for which inter-point distances are invariant for all local motions of the framework. De-

*Research started at the *Workshop on Geometric Aspects of Molecular Modeling*, organized by the second author at the Bellairs Research Institute of McGill University in Barbados in Jan. 2003 and partially supported by NSF grant CCR-0203224.

†Department of Computer Science, UNC Chapel Hill, CB 3175 Sitterson Hall, Chapel Hill, NC, USA, 27599–3175. snoeyink@cs.unc.edu. Partially supported by NSF grant 0086013.

‡Department of Computer Science, Smith College, Northampton, MA 01063, USA, streinu@cs.smith.edu. Partially supported by NSF grant CCF-0430990.

ciding rigidity or detecting rigid components for *generic frameworks* reduces to a purely combinatorial structure: Laman graphs. A *generically minimally rigid graph* or *Laman graph* on n vertices has exactly $2n-3$ edges, and subsets of $n' < n$ vertices span *at most* $2n'-3$ edges. Laman graphs are maximal independent sets of edges in the generic dimension 2 rigidity matroid [2], which guarantees that maximal sets on the same number of vertices have the same size and allows greedy algorithms to compute rigid components. For instance, the Pebble Game algorithm [6] runs in quadratic time in the worst case [1, 9], although better performance has been observed in practical applications [5].

We can apply the theory of generic rigidity to pointed pseudo-triangulations because the underlying graph of a pointed pseudo-triangulation is a *Laman graph with a generic embedding* [12]. However, the additional information of planarity and pointedness gives hope for even better running times.

Our main result is a *linear-time algorithm* to compute the rigid components of a pointed pseudo-triangulation mechanism. We then extend the algorithm to arbitrary pointed planar graphs.

2 Convex regions and rigid components

We define convex regions in pointed graphs. Then we show that the rigid components of pointed pseudo-triangulation mechanisms are the convex regions from which no edges have been removed. This sets up the algorithm to find rigid components – simply find the maximal convex regions.

A cycle in a graph is an alternating sequence of edges and vertices; a simple cycle does not repeat a vertex. A simple cycle in a pointed planar graph is *convex* if it can be oriented so that each cycle vertex has its large angle to the right. A convex cycle bounds a convex region, which is the region that doesn't contain the point at infinity. In a pointed graph, when convex regions intersect in more than one point, then their union is also convex; this will help us find maximal convex regions.

Lemma 1 *If two convex regions in a pointed planar graph intersect in more than one point, then their union is a convex region.*

Proof. Two convex regions that satisfy the lemma's hypotheses are either nested, and the lemma is trivially true, or have boundary cycles that intersect in two or more points. These points must be vertices of the planar graph, which have large angles to their right for both cycles, and therefore for their union. Thus, the cycle bounding the infinite face of the union of boundary cycles is convex. □

Pointed pseudo-triangulations can be constructed from any pointed graph by adding straight-line edges that preserve pointedness and planarity. We can extend previous observations on the edges in pseudo-triangulations to count edges [12]:

Lemma 2 *Any subgraph of a pointed pseudo-triangulation with v vertices has at most $2v-3$ edges, with equality attained by and only by convex regions.*

Laman's theorem [10, 2] says that these counts exactly characterize the generic minimally rigid graphs in the plane, giving as corollary:

Corollary 3 *An induced subgraph of a pointed pseudo-triangulation is rigid if and only if it is a convex region.*

3 Spiral walk to find maximal convex regions in pseudo-triangulation mechanisms

The observations of the previous section suggest a simple approach for finding the rigid components in a pointed graph: detect maximal convex subgraphs that do not include any deleted edges. For pointed pseudo-triangulation mechanisms, where a single edge is edge deleted from the convex hull, this is easy to do with a *spiral walk*. We focus on this important special case in this brief paper, and just sketch the extension.

We give a simple algorithm that identifies the maximal convex regions in time proportional to the number of edges on their boundaries. We use a standard winged-edge or quad-edge [3] data structure for storing connected planar subdivisions, which can access the edges around a vertex in cw or ccw order, and can access a vertex from any incident edge.

Recall that in a pointed graph, every vertex is incident to an angle larger than π. We call the edges bounding the large angle the *cw* and *ccw extreme edges* of the vertex, clockwise and counter-clockwise from the large angle. Any other edges are called *internal*. We augment the data structure with pointers from each vertex to its cw and ccw extreme edges.

We start the spiral walk on an edge that is known to bound a convex region. In the beginning, this can be any edge on the infinite face. When we reach a vertex, we continue walking on the ccw extreme edge. If this was the edge on which we reached the vertex, then we have detected a single-edged maximal convex region. Otherwise, we have made a left turn that keeps the large angle to the right, and when we return to some vertex v, we close a maximal convex region. In either case, we remove the detected convex region from the graph, then continue from v if any edge remains incident on v from the vertex where we left off. When this algorithm terminates, each edge has been assigned to some convex region.

We give more detail in Alg. 1, using Guibas and Stolfi's quad edge structure [3] to store connected com-

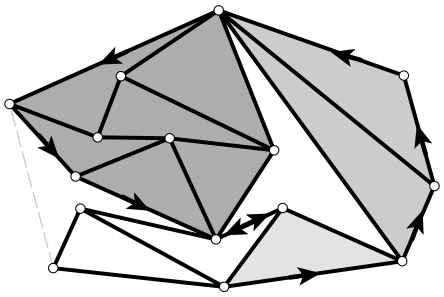

Figure 2: Spiral walk to find rigid components. First the edge is detected (double arrow), then the shaded convex pentagon, then quadrilateral, then triangle. Since we started at the lowest vertex, instead of the deleted edge, the remaining two components will be found by a second walk.

ponents of the embedded graph. We can use it to access, from a directed edge e, the reverse edge $e.Sym$, the edges in ccw order around the faces to the right, $e.Rnext$ and $e.Rprev$, and the origin and destination vertices, $e.Org$ and $e.Dest$. For details on the construction operator $\texttt{Splice}(a, b)$, see [3]. We add a $e.Mark$ field to mark edges as unmarked, visited, or deleted. These are initially unmarked. With each vertex v we store the ccw extreme edge $v.ccwExtr$ (or *null* if there is no incident edge). We use the abbreviation $e.ccwDest = e.Dest.ccwExtr$.

The algorithm maintains three invariants. First, the graph remains pointed at all times. Second, the marked edges constitute a spiral walk that starts on an infinite face and keeps the large angles to the right. Thus, when an edge is revisited, we have a convex region—we will show in a moment that detected convex regions are maximal. Third, every connected component that has not being considered has at least one edge from its outer face in the bag.

The second invariant makes it clear that the identified regions are convex, and the third that every edge is assigned to some convex region. What remains to show is that the identified regions are maximal, especially since we take the graph apart as we traverse it. We can do this by observing the order in which regions are identified.

Lemma 4 *The spiral walk identifies maximal convex regions in a pseudo-triangulation mechanism.*

Proof. Suppose, for the sake of deriving a contradiction, that cycle C is the first convex cycle identified by the spiral walk that is not maximal–that there is some convex region R that contains at least an edge of C. By Lemma 1 we know that if C intersects some convex region R in more than one point, then their union is convex. We can consider that the walk that found C began at infinity, therefore outside of R. For this walk to go

```
while ∃e ∈ B
    if e is already deleted, continue;
    otherwise, repeat
        repeat                                  /* Find a cycle */
            mark e visited;
            set e = e.ccwDest;
        until e is marked.

        set e' = e
        repeat                          /* Cut cycle from graph */
            Mark e' deleted;
            set s = e'.Sym;
            if s ≠ e'.ccwDest                   /* need to cut */
                Splice(s, e'.ccwDest.Rprev);
                Set e'.Dest.ccwExtr = s;
                Add s to B; /* Save edge on outer face */
            else                        /* no edge from e'.Dest */
                set e'.Dest.ccwExtr = null;
            endif
            set e' = e'.ccwDest;
        until e = e'.                   /* Report cycle from e */

        set e = e.Org.ccwExtr;
    until e = null
endwhile
```

Algorithm 1: Finding the maximal convex regions of a pointed pseudo-triangulation mechanism. The connected components are stored in a quadedge data structure. A bag of edges B, one per component, is maintained. Initially, there is only one component and the bag contains an edge from its outer face.

inside of R at a vertex v, the ccw extreme edge on R at v must have been removed earlier, which can happen only if R was removed earlier, and since R was maximal by choice of C, this would take away any edges of C in R. Thus C could not be found. This contradiction establishes the lemma. □

Theorem 5 *The rigid components of a pseudo-triangulation mechanism can be found in linear time.*

Proof. By the preceding Lemmas and Corollary, the algorithm correctly finds maximal convex regions, which are the rigid components. To analyze the running time, note that each edge is traversed at most twice. (In fact, edges inside convex regions are not traversed at all.) Thus, it runs in time proportional to the number of edges in the maximal convex cycles, which is certainly $O(n)$, since a pseudo-triangulation mechanism has exactly $2n - 4$ edges [12]. □

This algorithm can be extended to find rigid components for a mechanism formed by deleting an edge e from inside a pseudo-triangulation, as in Figure 3.

We must take care that our convex cycles do not enclose the deleted edge e. We modify the algorithm as follows: each endpoint that has deleted edge e as internal is split, making two pointed vertices. The edges that were neighbors of e become extreme edges. This avoids cycles that enclose e at the endpoints; we must still watch out for walks that circle the convex hull and contain e inside.

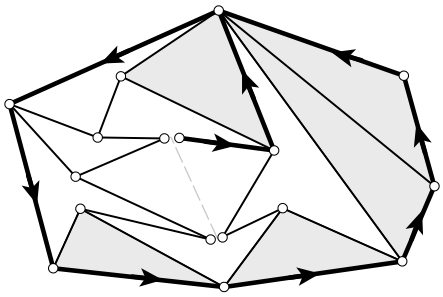

Figure 3: If we walked from inside, we might find the convex hull containing the deleted edge. Walking from pairs detects the single edge component instead.

Since some convex cycle must visit the endpoints of e, we can put the extreme edges for these endpoints into the initial bag in pairs. We can then walk in lock-step from both edges of a pair, one going cw and one ccw, stopping whichever walk reaches the initial direction of the other. We detect convex regions by revisited vertices, as before; when we remove a convex region, we put extreme edge pairs in the bag for any vertex that gets a new extreme edge. We omit precise details due to space constraints.

4 Conclusions

The main application of this result is a more robust implementation and a substantial reduction in numerical computation for the solution to the Carpenter's Rule problem based on pseudo-triangulation mechanisms [12, 13]. Since the vertices inside a rigid component do not move relative to each other, the extreme edge pairs for such vertices need not be tested for alignment. It remains open whether this simplification leads to an asymptotic reduction in the number of alignment events of the algorithm.

The simple algorithm we described makes use of the strong constraints of pointedness and planarity to detect rigid components in better than the quadratic time of the general case. Natural open questions remain (see also [13]):

Open Problem 1 *Is there a subquadratic time algorithm for rigid component detection for a* planar *(non-crossing) mechanism, not necessarily embedded as a pseudo-triangulation mechanism?*

Open Problem 2 *What is the complexity of deciding whether a* planar *(non-crossing) graph with* $2n-3$ *edges is Laman? Can this be done in subquadratic time?*

References

[1] A. Berg and T. Jordán. Algorithms for graph rigidity and scene analysis. In G. D. Battista and U. Zwick, eds, *Proc. 11th European Symp. on Algorithms (ESA)*, vol. 2832 of *Lect. Notes in Comp. Science*, pp. 78–89. Springer, 2003.

[2] J. Graver, B. Servatius, and H. Servatius. *Combinatorial Rigidity*. Graduate Studies in Mathematics vol. 2. American Mathematical Society, 1993.

[3] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, 4(2):74–123, April 1985.

[4] R. Haas, D. Orden, G. Rote, F. Santos, B. Servatius, H. Servatius, D. Souvaine, I. Streinu, and W. Whiteley. Planar minimally rigid graphs and pseudo-triangulations. *Comp. Geometry: Theory and Applic.*, pages 31–61, May 2005.

[5] D. Jacobs. personal communication, 2003.

[6] D. J. Jacobs and B. Hendrickson. An algorithm for two dimensional rigidity percolation: The pebble game. *J. Comput. Phys.*, 137:346–365, 1997.

[7] D. J. Jacobs, L. A. Kuhn, and M. F. Thorpe. Flexible and rigid regions in proteins. In Thorpe and Duxbury, editors, *Rigidity Theory and Applications*, pages 357–384. Kluwer Academic/Plenum Press, 1999.

[8] D. J. Jacobs and M. F. Thorpe. Generic rigidity percolation: the pebble game. *Phys. Rev. Lett.*, 75:4061–4054, 1995.

[9] A. Lee, I. Streinu, and L. Theran. Finding and maintaining rigid components. These proceedings, Canadian Conf. Comp. Geom.2005.

[10] G. Laman. On graphs and rigidity of plane skeletal structures. *J. Engrg. Math.*, 4:331–340, 1970.

[11] A. J. Rader, B. M. Hespenheide, L. A. Kuhn, and M. F. Thorpe. Protein unfolding: Rigidity lost. *PNAS*, 22(6):3540–3545, March 19 2002.

[12] I. Streinu. A combinatorial approach to planar non-colliding robot arm motion planning. In *ACM/IEEE Symp. on Found. of Comp. Science (FOCS)*, pages 443–453, 2000.

[13] I. Streinu. Combinatorial roadmaps in configuration spaces of simple planar polygons. In S. Basu and L. Gonzalez-Vega, eds, *Proc. DIMACS Workshop Algorithmic and Quantitative Aspects of Real Algebraic Geometry in Math. and Comp. Science*, pp. 181–206. DIMACS, 2003.