Revisiting Decomposition Analysis of Geometric Constraint Graphs

R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, J. Vilaplana-Pastó

Universitat Politècnica de Catalunya
Departament de Llenguatges i Sistemes Informàtics
Av. Diagonal 647, 8a, E–08028 Barcelona

e-mail: `[robert, tonis, sebas, josep]@lsi.upc.es`

Abstract

Geometric problems defined by constraints can be represented by geometric constraint graphs whose nodes are geometric elements and whose arcs represent geometric constraints. Reduction and decomposition are techniques commonly used to analyze geometric constraint graphs in geometric constraint solving.

In this paper we first introduce the concept of *deficit* of a constraint graph. Then we give a new formalization of the decomposition algorithm due to Owen. This new formalization is based on preserving the deficit rather than on computing triconnected components of the graph and is simpler. Finally we apply tree decompositions to prove that the class of problems solved by the formalizations studied here and other formalizations reported in the literature is the same.

Keywords Constraint solving, geometric constraints, graph-based constraint solving.

1 Introduction

Geometric problems defined by constraints can be represented by geometric constraint graphs whose nodes are geometric elements and whose arcs represent geometric constraints.

For application with potentially large constraints systems, the efficiency of the algorithms for solving the system at hand is an important issue. To decide on the suitability of a given constraint solving method, its correctness must be proved and the class of problems the method can solve should be characterized.

Many attempts to provide general, powerful and efficient methods for solving systems of geometric constraints have been reported in the literature. For an extensive review in geometric constraint solving refer to Fudos [6] and Durand [3].

Among the existing methods we focus on two techniques commonly used to analyze geometric constraint graphs in geometric constraint solving, generically known as *decomposition* and *reduction*, respectively. More specifically we are interested in decomposition and reduction where the analysis is based on a direct geometric interpretation. There are other approaches. See for example Hoffmann *et al.* [7] for a flow-based decomposition algorithm.

In [13], Owen described a top-down algorithm for computing a decomposition of an arbitrary graph. The algorithm recursively splits the graph into split components, [8]. The algorithm terminates when the graphs cannot be split further. At the end of the analysis the original graph has been decomposed into a set of triangles.

Fudos and Hoffmann, [6], reported on a graph-constructive approach to solving systems of geometric constraints. The method is based on an analysis of the constraint graph that derives a sequence of construction steps that sequentially places the geometric elements in the problem with respect to each other. The analysis has two parts. The first part is a bottom-up reduction analysis where each step in the sequence corresponds to positioning three rigid geometric bodies that pairwise share a geometric element, point or line. The second part is a top-down decomposition analysis that produces a sequence of decompositions that correspond to a reverse sequence of rigid geometric bodies.

In this paper we reformulate the algorithm reported by Owen in [13] to solving geometric constraint problems based on the decomposition analysis of the constraint graph. First we introduce the concept of *deficit* associated with a constraint graph. The deficit measures the distance between a given constraint graph and a well-constrained graph induced by the same set of nodes. The deficit allows us to avoid the need for computing triconnected components yields a simpler algorithm both conceptually and from a computational runtime point of view. Then we recall the *tree decomposition* of a constraint graph, a tool that has is useful to conceptually analyze constraint graphs. Finally tree decompositions are applied to characterize the class of problems solved by the decomposition analysis studied here and to prove that different formalizations solve the same class of problems.

Section 2 reviews basic concepts from graph theory and geometric constraint graphs. Section 3 deals with decomposition analysis. First we recall Owen's algorithm, then we present the new formalization of the algorithm. Section 4 presents the tree decomposition of a constraint graph. Section 5 is devoted to characterize the class of problems solved by the decomposition analysis studied here and discusses the equivalence of different formalizations. We close with a brief summary in Section 6.

2 Preliminaries

In this section we recall basic terminology of graph theory, the concept of geometric constraint graph associated to a geometric problem defined by constraints, and some definitions related to geometric constraint graphs.

2.1 Graph Concepts

First we recall some basic terminology of graph theory that will be used in the rest of the paper. For an extensive treatment see [2] and [8].

A graph $G = (V, E)$ is said to be *connected* if every vertex is connected to every other node by at least one path of edges. We say that a node a of a connected graph G is an *articulation node* if by removing a, the graph splits into two or more disconnected subgraphs. If a is an articulation node in G, then there are two vertices

u and v different from a such that a is on every path connecting u and v.

A graph with no articulation vertices is called *biconnected*. If u and v are arbitrary different vertices of a biconnected graph G, then there are at least two different paths in G connecting them. A connected graph can be uniquely decomposed into biconnected components by splitting it at separation vertices. Aho *et al.*, [1], reported a depth first algorithm that efficiently computes such a decomposition.

Let a and b be two vertices in a biconnected graph G. The edges of G can be divided into *separation classes* $E_1, E_2, \ldots, E_n$ defined as follows, [8]. Two edges are in the same separation class E_i if there is a path using both edges and not containing a or b except, possibly, as endpoints. If the two vertices a and b divide the edges into more than two separation classes, then the pair $\{a, b\}$ is a *separation pair* (*articulation pair*) of G. Moreover, if $\{a, b\}$ divides the edges into two separation classes, each containing more than one edge, then $\{a, b\}$ is also a separation pair.

A *triconnected graph* is a graph with more that two vertices with no separation pairs. In a triconnected graph there are at least three disjoint paths between every pair of non adjacent vertices.

Let $\{a, b\}$ be a separation pair in the graph G that induces the separation classes $E_1, E_2, \ldots, E_n$. Let $E' = \bigcup_{i=1}^{m} E_i$ and $E'' = \bigcup_{i=m+1}^{n} E_i$ such that $|E'| \geq 2$ and $|E''| \geq 2$. Then we will refer to the graphs $G' = (V(E'), E')$ and $G'' = (V(E''), E'')$ as the *separating graphs* of G. The graphs

$$G_1 = (V(E'), E' \cup \{(a, b)\})$$

and

$$G_2 = (V(E''), E'' \cup \{(a, b)\})$$

are called *split graphs* of G. The added edge (a, b) is labeled to denote the split and is called a *virtual edge*. Assume that the graph G and its split graphs are recursively split until obtaining graphs that cannot be split further. The set of these graphs defines the set of *split components* of G. Note that the split components are triconnected graphs and that by merging the split components we recover the original graph.

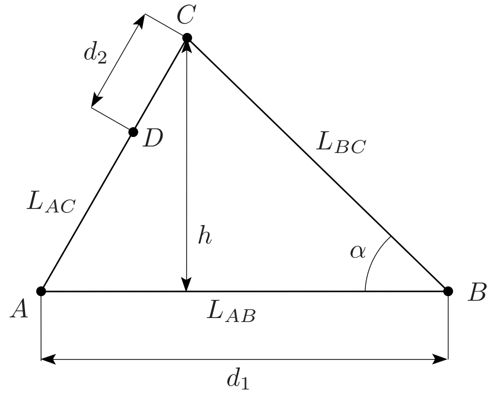
Hopcroft and Tarjan, [8], and Miller and Ramachandran, [12], reported on algorithms to efficiently compute separating graphs and split components of a graph.

2.2 Geometric Constraint Solving and Graphs

A geometric constraint problem is defined by giving a set of geometric elements like points, lines, line segments, circles and circular arcs, along with a set of relationships, called constraints, like distance, angle, incidence and tangency between any two geometric elements. As explained by Fudos, [4] and by Mata, [11], we may transform the geometric constraint problem into one where only points and lines with pairwise distance and angle constraints need to be considered.

The geometric constraint problem can be coded as a *constraint graph* $G = (V, E)$, where the graph vertices V are the geometric elements and the graph edges E are the geometric constraints, [13, 6]. Figure 1 shows a geometric problem defined by constraints and the geometric constraint graph associated.

A necessary condition for a geometric constraint problem to be solvable is that the associated constraint graph must be wellconstrained. Combinatorial properties of wellconstrained graphs have been characterized by Laman, [10]. Technically, the notion of wellconstrained graph can be formalized as follows, [6].



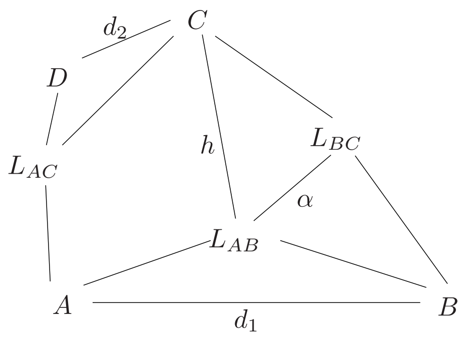


Figure 1: A geometric constraint problem and associated constraint graph.

Definition 2.1 *Let $G = (V, E)$ be a geometric constraint graph.*

1. *G is* structurally over-constrained *if there is an induced subgraph with $m \leq |V|$ vertices and more than $2m - 3$ edges.*
2. *G is* structurally under-constrained *if it is not structurally over-constrained and $|E| < 2\,|V| - 3$.*
3. *G is* structurally well-constrained *if it is not structurally over-constrained and $|E| = 2\,|V| - 3$.*

3 Decomposition Analysis

First we briefly recall the Owen's algorithm. Then we give a new formalization for the Owen's algorithm which is simpler and that will be used in the following sections.

3.1 Owen's Algorithm

Owen in [13] introduced a geometric constraint solving technique based on a top-down analysis of the geometric constraint graph associated with a geometric problem.

The algorithm has two steps. In a first step, the algorithm computes the set of split components S of the given graph G, [8]. These split components are either triangles or complex triconnected graphs, that is, graphs with more than three edges. As computed, the complex split components are no further decomposable. To overcome this problem, in a second step the complex split components are transformed, if possible, by removing from the graph one of the virtual edges introduced in the first step. Note that virtual edges are always incident to separation pairs.

Then the first step is recursively applied to the transformed split components. The algorithm terminates when the graphs cannot be split further. If any triconnected graph with more than three vertices remains, the problem cannot be solved quadratically.

At the end of the analysis, the original graph has been decomposed into a set of triangles whose edges are either original edges or virtual edges.

If function `SplitComponents(G)` computes the split components of G, function `Reducible(g)` checks whether a split component should be further subdivided, and function `Reduce(g)` removes unneeded virtual edges of graph g, Owen's algorithm can be written as shown in Figure 2.

```
func Owen($G$) ret $S$
  $SC$ := SplitComponents($G$)
  $S$ := $\emptyset$
  foreach $g$ in $SC$ do
    if Reducible($g$) then
      $S$ := $S \cup$ Owen(Reduce($g$))
    else
      $S$ := $S \cup \{g\}$
    fi
  done
  return $S$
end
```

Figure 2: Owen's analysis algorithm.

The analysis process followed by Owen's analysis algorithm is illustrated in Figure 3. Virtual edges are shown in dashed lines.

3.2 The New Formalization

To decompose a graph, Owen's method uses the algorithm for finding triconnected components reported by Hopcroft and Tarjan in [8]. which is based on preserving graph connectivity. As a result, the split components generated by the decomposition include extra virtual edges. To recursively apply the decomposition process, Owen's algorithm must remove these extra virtual edges.

In what follows we will present an algorithm to decompose a constraint graph in triconnected graphs with exactly three vertices, that is, triangles. The algorithm is based on a divide and conquer strategy which preserves the constraint graph property of being wellconstrained. The resulting algorithm is conceptually simple and easy to implement.

As in [13] and [6], the algorithm will be based on subdividing the constraint graph into two separating graphs induced by a separation pair. With the aim of clearly stating a subdivision criterion, we start by giving some definitions and deriving properties which relate wellconstrained graphs with their separating graphs.

Definition 3.1 *Let $G = (V, E)$ be a geometric constraint graph. We define the* `Deficit` *function associated with G by*

$$\textit{Deficit}(G) = (2|V| - 3) - |E|$$

The function `Deficit` computes the difference between the number of edges needed for a constraint graph to be wellconstrained and its actual number of edges. Note that if G is not overconstrained, Deficit$(G) \geq 0$

Lemma 3.2 *Let G be a constraint graph and G' and G'' separating graphs. Then*

$$\textit{Deficit}(G) = \textit{Deficit}(G') + \textit{Deficit}(G'') - 1$$

Proof
By definition, Deficit$(G) = (2|V| - 3) - |E|)$. Since G' and G'' are separation graphs of G, then $|V| = |V'| + |V''| - 2$ and $|E| = |E'| + |E''|$. Therefore,

$$\begin{aligned} \text{Deficit}(G) &= 2(|V'| + |V''| - 2) - 3 - (|E'| + |E''|) \\ &= (2|V'| - 3 - |E'|) + (2|V''| - 3 - |E''|) - 1 \\ &= \text{Deficit}(G') + \text{Deficit}(G') - 1 \end{aligned}$$

□

Lemma 3.3 *Let G be a wellconstrained graph and G' and G'' separating graphs. Then if Deficit(G') > Deficit(G''), G' is underconstrained and G'' is wellconstrained.*

Proof
Since G is wellconstrained, Deficit$(G) = 0$ and separation graphs, G' and G'', are not overconstrained, that is, Deficit$(G') \geq 0$ and Deficit$(G'') \geq 0$. From Lemma 3.2 Deficit$(G) =$ Deficit$(G') +$ Deficit$(G'') - 1$. Thus Deficit$(G') +$ Deficit$(G'') = 1$. Then, either Deficit$(G') = 1$ and Deficit$(G'') = 0$, which means that G' is underconstrained and G'' wellconstrained or vice versa. □

Definition 3.4 *Let G be a wellconstrained constraint graph and G' and G'' separating graphs. The* modified split graphs, *G_1 and G_2, of G are defined as follows. If Deficit(G') > Deficit(G'') then*

$$G_1 = (V(E'), E' \cup \{(a, b)\}) \quad \textit{and} \quad G_2 = G''$$

otherwise

$$G_1 = G' \quad \textit{and} \quad G_2 = (V(E''), E'' \cup \{(a, b)\})$$

Lemma 3.5 *Let $G = (V, E)$ be a constraint graph and, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be modified split graphs. Then Deficit$(G) =$ Deficit$(G_1) +$ Deficit(G_2).*

Proof
Now $|E| = |E_1| + |E_2| - 1$. Apply proof of Lemma 3.2. □

Definition 3.6 *An* s-tree *is a binary tree such that:*

1. *the root is a constraint graph G,*
2. *for each node in G the root of their sons are the modified split graphs S_1 and S_2 of G, and*
3. *the leaves are either triangles or triconnected graphs.*

Let `Triconnected(`G`)` be a function that tests whether a graph has a separation pair, `SeparatingGraphs(`G`)` a function that computes the separating graphs of G, (Recall that separating graphs do not include virtual edges), and `AddVirtualEdge(`G`)` a function that adds a virtual edge incident to the separation pair used to compute the split graph G. Then the decomposition analysis algorithm based on preserving deficits of graphs can be written as shown in Figure 4.

The input to the algorithm is a graph G associated to a geometric constraint problem. The output is a s-tree S whose root is G.

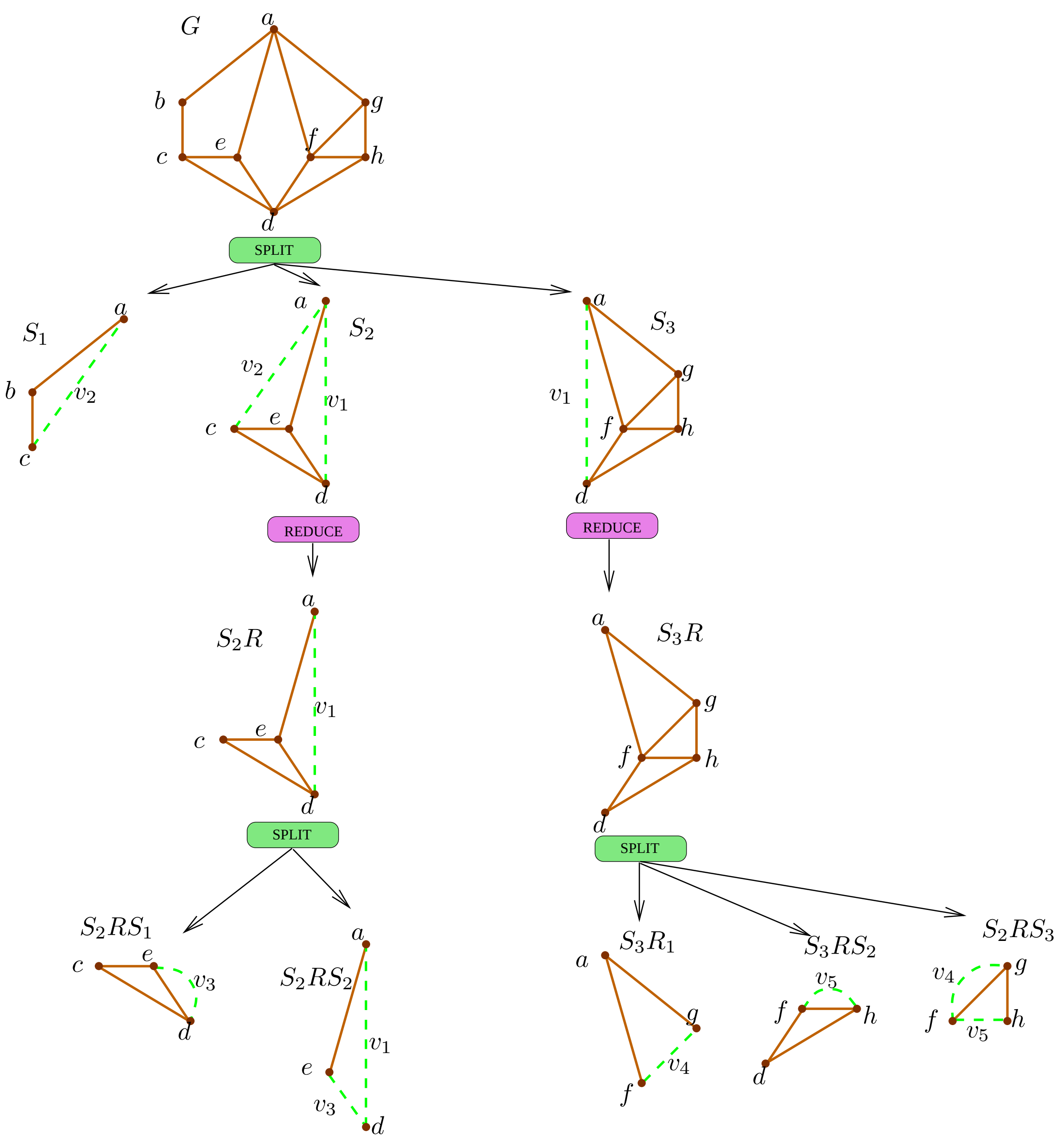


Figure 3: Owen’s algorithm computation applied to an example graph.

```
func Analysis($G$) ret $S$
  if Triconnected($G$) then
    $S$ := BinaryTree($G$, nullTree, nullTree)
  else
    $G_1$,$G_2$ := SeparatingGraphs($G$)
    if Deficit($G_1$) $>$ Deficit($G_2$) then
      $G_1$ := AddVirtualEdge($G_1$)
    else
      $G_2$ := AddVirtualEdge($G_2$)
    fi
    $S$ := BinaryTree($G$, Analysis($G_1$),
                         Analysis($G_2$))
  fi
  return $S$
end
```

Figure 4: New algorithm for decomposition analysis.

Note that if G represents a wellconstrained solvable problem, the resulting s-tree decomposes G into triangles.

Figure 5 illustrates the behaviour of the new decomposition analysis algorithm applied to the example graph in Figure 3. Note that now only those virtual edges that are strictly necessary to keep the deficit property are included in the modified split graphs, therefore avoiding the need for graph transformation.

When one of the separation classes is a single edge, it is incident to the vertices in the separation pair. In this case we prove the following result.

Lemma 3.7 *Let $G = (V, E)$ be a wellconstrained geometric constraint graph, $\{a, b\}$ a separation pair such that $(a, b) \in E$. Then the separation graph which contains the edge (a, b) is wellconstrained and no virtual edge is added.*

Proof
Let $\{a, b\}$ be the separation pair in the graph $G = (V, E)$ and $E_1, \ldots, E_{n-1}, E_s$ be the separation classes, where E_s contains just the edge (a, b). Let $\mathcal{E}' = \bigcup_{i=1}^{m} E_i$ and $\mathcal{E}'' = \bigcup_{i=m+1}^{n-1} E_i$. such that $|E'| \geq 2$ and $|E''| \geq 2$. We have

$$|E| = |\mathcal{E}'| + |\mathcal{E}''| + |E_s| = |\mathcal{E}'| + |\mathcal{E}''| + 1$$

$$|V| = |V(\mathcal{E}')| + |V(\mathcal{E}'')| - 2$$

G wellconstrained means that $2|V| - 3 - |E| = 0$. Substituting $|E|$ and $|V|$ by the expressions above and rearranging terms

$$(2|V(\mathcal{E}')| - 3 - |\mathcal{E}'|) + (2|V(\mathcal{E}'')| - 3 - |\mathcal{E}''|) - 2 = 0$$

There are two different situations. First let

$$(2|V(\mathcal{E}')| - 3 - |\mathcal{E}'|) = (2|V(\mathcal{E}'')| - 3 - |\mathcal{E}''|) = 1$$

Since classes E_i are grouped arbitrarily, assume that $E' = \mathcal{E}' \cup E_s$. Then $V(E') = V(\mathcal{E}')$ and $|E'| = |\mathcal{E}'| + 1$. The deficit of the separation graph G' is

$$\begin{aligned}(2|V(E')| - 3 - |E'|) &= 2|V(\mathcal{E}')| - 3 - (|\mathcal{E}'| + 1)\\ &= 2|V(\mathcal{E}')| - 3 - |\mathcal{E}'| - 1\\ &= 0\end{aligned}$$

Therefore the separation graph G' contains edge (a, b), is wellconstrained and, since Deficit$(G') <$ Deficit(G''), no virtual edge will be added to it.

For a contradiction and without loss of generality, let $(2|V(\mathcal{E}')| - 3 - |\mathcal{E}'|) = 0$, that is, G' is wellconstrained. Let $(2|V(\mathcal{E}'')| - 3 - |\mathcal{E}''|) = 2$. and assume again $E' = \mathcal{E}' \cup E_s$. This would result in the separation graph $G' \subset G$ being overconstrained, that is G would be overconstrained which is a contradiction. □

Lemmas 3.2, 3.3 and 3.5 along with Lemma 3.7 prove the correctness of the algorithm presented in this section.

3.3 Subdivision Pattern

The algorithm given in the previous section analyzes a constraint graph by decomposing it into two split graphs induced by a separation pair. However, there is nothing essential in this subdivision method.

Todd, [14], reported on a method where graphs are subdivided by isolating vertices of degree two from their neighbors. In fact, this is a particular case of decomposing through separation pairs. Figure 6 illustrates a graph with a degree two vertex v. Note that its neighbors, a and b, are a separation pair. This subdivision method is rather limited but can be satisfactorily combined with other methods, like those in [8] or in [12], to compute more general graph subdivisions.

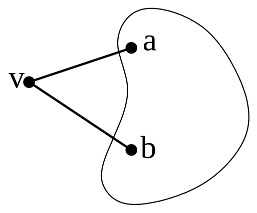


Figure 6: Graph with a degree two vertex v.

Another method subdivides a graph into three subgraphs by selecting three vertices such that by removing them the graph splits into three connected components. See Figure 7. We do not know any efficient algorithm to select the three vertices but they can be always computed by using a brute force approach.

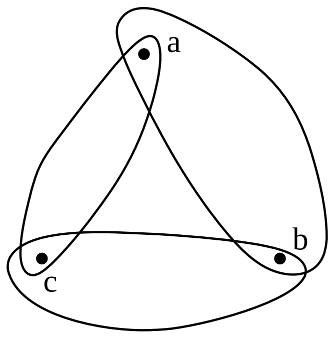


Figure 7: Subdividing a graph into three subgraph by using three vertices a, b and c.

4 Tree Decomposition

In this section first we define the concept of *set decomposition* that refers to a way of partitioning a given abstract set. Then we define the concept of *tree decomposition* of a graph, a tool that we will use in Section 5.

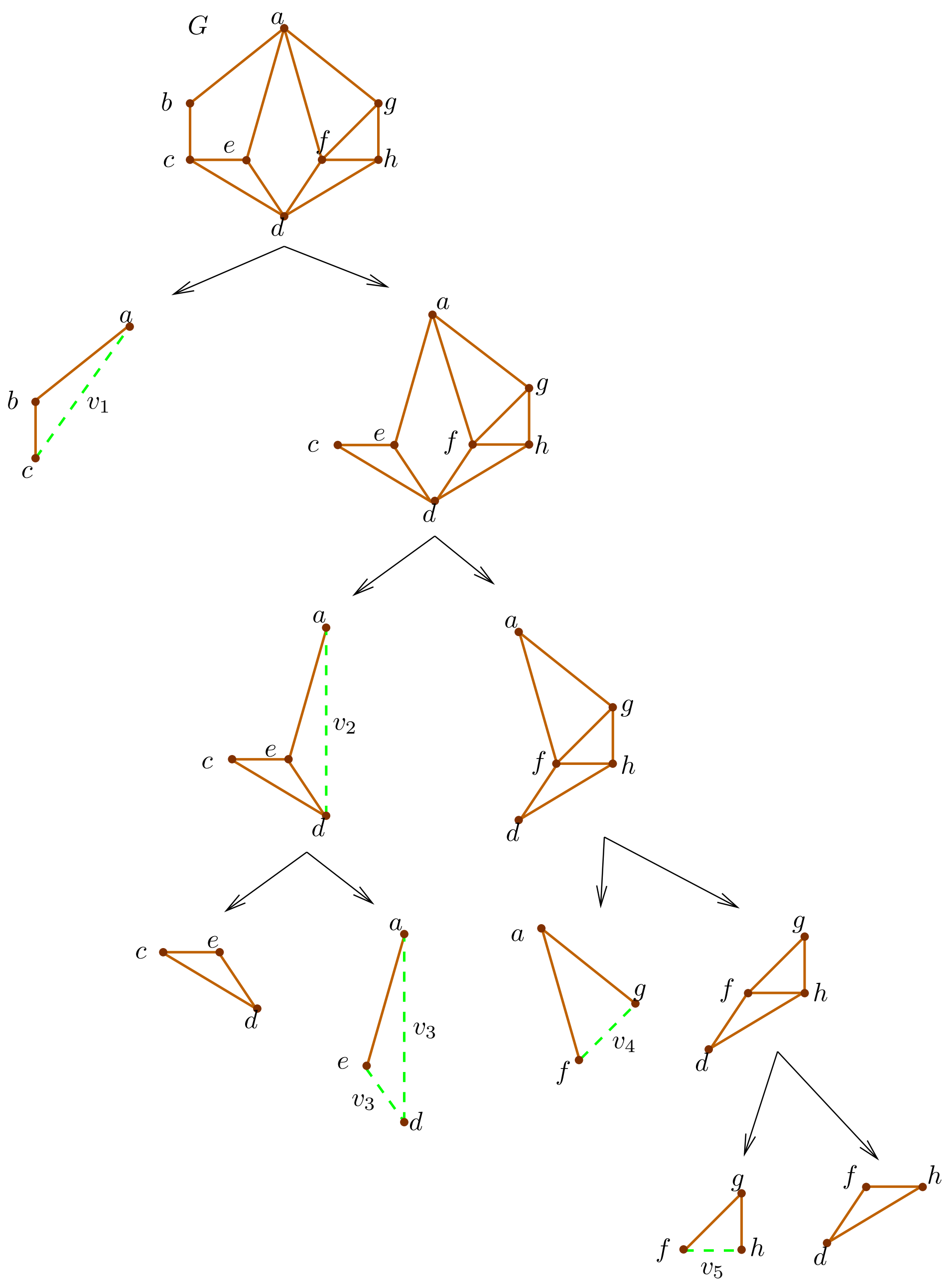


Figure 5: Decomposition analysis generated by the new algorithm on the example graph in Figue 3.

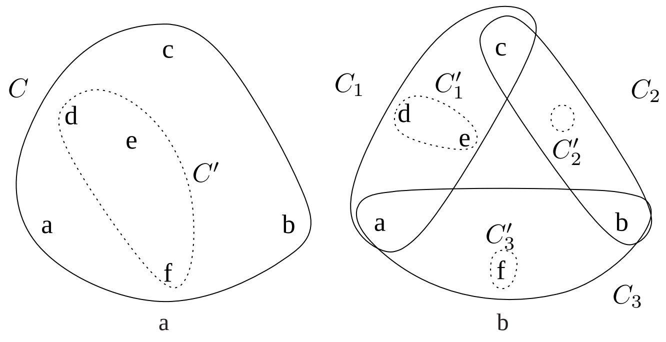


Figure 8: a) A set C. b) A set decomposition of C.

Definition 4.1 *Let C be a set with, at least, three different members, say a, b, c. Let $\{C_1, C_2, C_3\}$ be three subsets of C. We say that $\{C_1, C_2, C_3\}$ is a* set decomposition *of C if*

1. *$C_1 \cup C_2 \cup C_3 = C$,*
2. *$C_1 \cap C_2 = \{a\}$,*
3. *$C_1 \cap C_3 = \{b\}$ and*
4. *$C_2 \cap C_3 = \{c\}$.*

We say that $\{a, b, c\}$ are the active elements *of the set decomposition.*

Figure 8 shows a set and a possible set decomposition. Next we define the concept of set decomposition of a graph, illustrated in Figure 9.

Definition 4.2 *Let $G = (V, E)$ be a graph. Let $V(e)$ denote the vertices in V that are the endpoints of edge $e \in E$. Let $\{V_1, V_2, V_3\}$ be three subsets of V. Then $\{V_1, V_2, V_3\}$ is a* set decomposition *of G if it is a set decomposition of V and for every edge e in E, $V(e) \subseteq V_i$ for some i, $1 \leq i \leq 3$.*

Roughly speaking, a set decomposition of a graph $G = (V, E)$, is a set decomposition of the set of vertices V such that does not *break* any edge in E. Figure 10 shows a graph and a set decomposition that is not a set decomposition of the graph because vertices incident to edge (e, b) does not belong to any set in the partition.

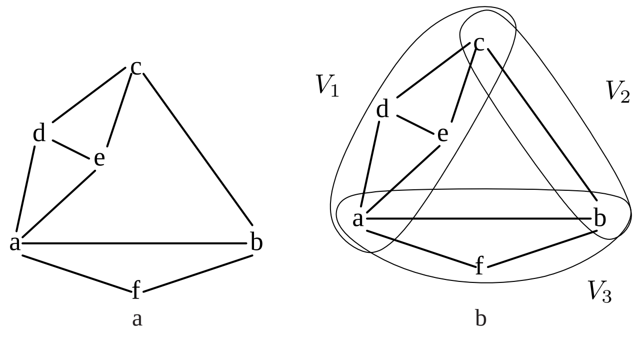


Figure 9: a) Graph. b) Set decomposition of the graph.

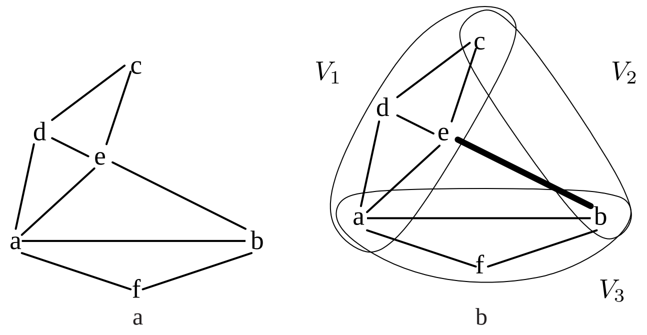


Figure 10: a) Graph. b) Set decomposition with a broken edge.

Lemma 4.3 *Let $\{V_1, V_2, V_3\}$ be a set decomposition of a graph G and let $V_1 \cap V_2 = \{a\}$ and $V_1 \cap V_3 = \{b\}$. If $|V_1| > 2$, then $\{a, b\}$ is a separation pair of G.*

Proof
The subgraphs of G induced by V_i, for $1 \leq i \leq 3$, have disjoint sets of edges. By Definition 4.1 $V_1 \cap (V_2 \cup V_3) = \{a, b\}$. Thus, removing $\{a, b\}$ disconnects G. Therefore $\{a, b\}$ is a separation pair. $\square$

To close this section, we define the concept of *tree decomposition* of a graph.

Definition 4.4 *Let $G = (V, E)$ be a graph. A 3-ary tree T is a* tree decomposition *of G if*

1. *V is the root of T,*
2. *Each vertex $V' \subseteq V$ of T is the father of exactly three nodes, say $\{V'_1, V'_2, V'_3\}$, which are a set decomposition of the subgraph of G induced by V', and*
3. *Each leaf node contains exactly two vertices of V.*

A graph for which there is a tree decomposition is a *tree decomposable* graph. Figure 11 shows a collection of set decompositions recursively generated for the tree decomposable graph of Figure 9. The corresponding tree decomposition is shown in Figure 12.

By Definition 4.4, all leaves of a tree decomposition T of a graph G have cardinality two.

5 Domain of Constructive Geometric Constraint Solving Techniques

Joan-Arinyo *et al.* showed in [9] that the class of tree decomposable graphs characterizes the domain of two constructive geometric constraint solving techniques: reduction and decomposition analysis as described respectively in [6] and [6, 9]. Here, we will see that the decomposition analysis studied in Section 3 can be characterized also by the existence of a tree decomposition and that it has the same domain as the reduction and decomposition analysis above mentioned.

In what follows we will only consider constraint graphs G associated with wellconstrained problems. In these conditions, *s-trees* are binary trees whose root is G, the other nodes are separation graphs with respect to some separation pair of the parent node and,

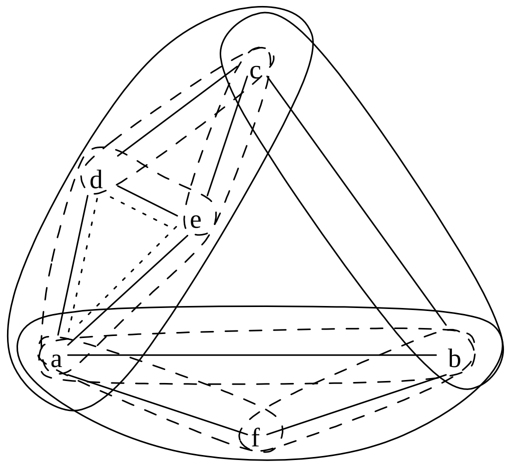


Figure 11: Collection of set decompositions of the graph in Figure 9.

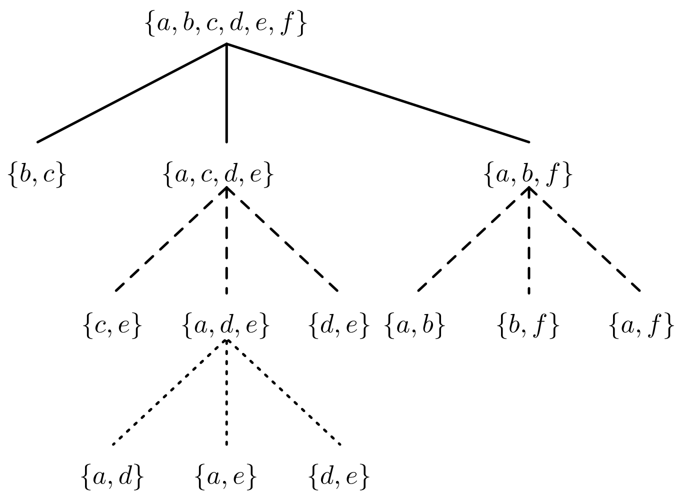


Figure 12: Tree decomposition of the graph in Figure 11.

the leaves are triangles or triconnected graphs with no articulation pairs.

Definition 5.1 *We say that a constraint graph G is* s-tree decomposable *if there is a s-tree such that its root is G and all its leaves are triangles.*

According to the number of virtual edges in the triangles in the leaf nodes of an s-tree, we classify them in four different types. See the second column in Table 1.

5.1 Domain Characterization

To characterize the decomposition analysis studied in Section 3 we prove two lemmas.

Lemma 5.2 *If a graph G is tree decomposable, then G is s-tree decomposable.*

Proof
Assume that T is a tree decomposition of G. We shall proceed by induction on the structure of T. Refer to Table 1.

Induction base: Let $G = (V, E)$ be a graph such that $V = \{a, b, c\}$ and $E = \{(a, b), (a, c), (b, c)\}$. The tree T in the third column of Table 1 is a tree decomposition of G. Then the tree in the fourth column is a s-tree whose root is a graph $G = G_0$ with just one node representing the triangle $\{a, b, c\}$.

Induction hypothesis: Let G' be a subgraph of G. If G' is tree decomposable then G' is s-tree decomposable.

Induction step: If $\{C_1, C_2, C_3\}$ is a set decomposition of C and $\{a, b, c\}$ the active elements, we have that $C' = C - \{a, b, c\}$, $C'_1 = C_1 - \{a, b\}$, $C'_2 = C_2 - \{a, c\}$ and $C'_3 = C_3 - \{b, c\}$.

Let G be a graph and T a tree decomposition of G such that its root is $\{a, b, c\} \cup C'$ and the roots of its subtrees are $\{a, b\} \cup C'_1$, $\{a, c\} \cup C'_2$ and $\{b, c\} \cup C'_3$.

Assume that $C'_1 \neq \emptyset$ and $C'_2 = C'_3 = \emptyset$. Let G'_1 be the subgraph induce by $\{a, b\} \cup C'_1$ in G. Let T_1 be the tree decomposition of G'_1.

By Lemma 4.3, (a, b) is a separation pair of G thus G'_1 is a separation graph of G. Build the other separation graph G_2 as the graph $G_2 = (V_2, E_2)$ with $V_2 = \{a, b, c\}$ and $E_2 = \{(a, c), (b, c)\}$. By Definition 2.1, G_2 is underconstrained, thus by Lemma 3.3, G'_1 is wellconstrained.

Now build the modified split graphs of G as $G_0 = (V_2, E_2 \cup \{(a, b)\})$ and G'_1.

Since G'_1 is tree decomposable, by the induction hypothesis it is s-tree decomposable. Therefore there is a s-tree, say S'_1, whose root is G'_1. Hence the binary tree whose root is G and whose subtrees are G_0 and S'_1 is a s-tree. Therefore G is s-tree decomposable.

Applying the same procedure for cases $C'_2 \neq \emptyset$ and $C'_3 \neq \emptyset$ completes the proof. □

If function `ComputeTriangle(`T`)` computes the triangle associated with a node of a tree decomposition, and function `MergeGraphs(`G_1, G_2`)` rebuilds a graph from its modified split graphs, Figure 13 shows an algorithm that, based on Lemma 5.2, computes a s-tree S from a tree decomposition T of a graph G.

Lemma 5.3 *If a graph G is s-tree decomposable, then G is tree decomposable.*

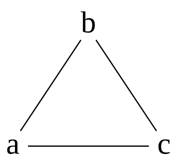
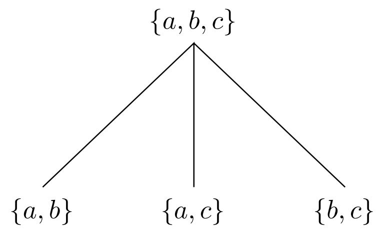
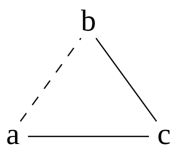
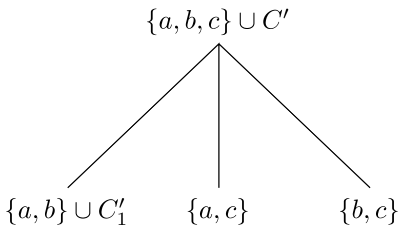
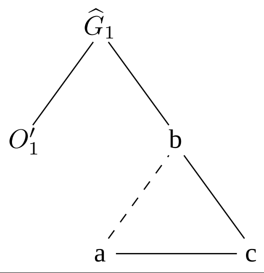
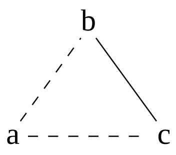
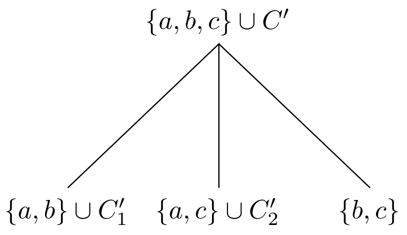
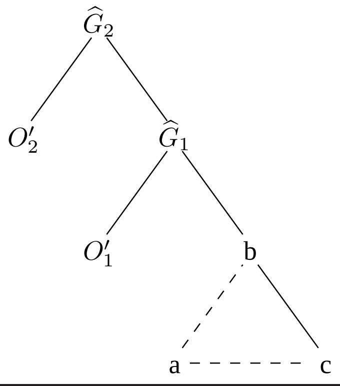
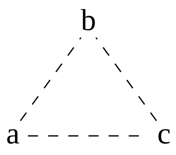
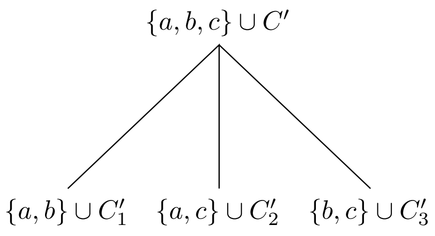
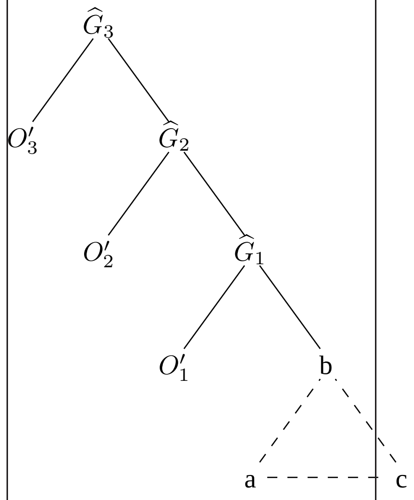
n	G_0	*Tree-decomposition* T	*S-tree* S
0			G_0
1		 $\{a,b,c\} \cup C'$ $\{a,b\} \cup C'_1$ $\{a,c\}$ $\{b,c\}$	
2		 $\{a,b,c\} \cup C'$ $\{a,b\} \cup C'_1$ $\{a,c\} \cup C'_2$ $\{b,c\}$	 $\widehat{G}_2$ O'_2 $\widehat{G}_1$ O'_1 b a c
3		 $\{a,b,c\} \cup C'$ $\{a,b\} \cup C'_1$ $\{a,c\} \cup C'_2$ $\{b,c\} \cup C'_3$	 $\widehat{G}_3$ O'_3 $\widehat{G}_2$ O'_2 $\widehat{G}_1$ O'_1 b a c

Table 1: Types of interior nodes in a tree decomposition and the equivalent s-tree decomposition.

```
func FromTreeToS-Tree($T$) ret $S$
  $G_0$ := ComputeTriangle($T$)
  $S$ := BinaryTree($G_0$, NullTree, NullTree)
  $n$ := NumberOfVirtualEdges($G_0$)
  for $j$ in 1 to $n$ do
    $T_j$ := Subtree($T, j$)
    $S'_j$ := FromTreeToS-Tree($T_j$)
    $G'_j$ := Root($S'_j$)
    $G_j$ := MergeGraphs($G'_j, G_{j-1}$)
    $S$ := BinaryTree($G_j, S'_j, S$)
  end
  return $S$
end
```

Figure 13: Computing a s-tree S from a tree decomposition T.

Proof
Assume that S is a s-tree whose root is G. Again we shall proceed by induction on the structure of S. Refer to Table 1.

Induction base: Let $G = (V, E)$ be a graph such that $V = \{a, b, c\}$ and $E = \{(a, b), (a, c), (b, c)\}$. The s-tree S of G is that given in the fourth column of Table 1. Then the tree given in the third column is a tree decomposition of G.

Induction hypothesis: Let G' be a subgraph of G. If G' is s-tree decomposable then G' is tree decomposable.

Induction step: Let S be a s-tree whose root is G and whose subtrees are G_0 and S'_1.

Assume that $G_0 = (V_0, E_0)$ with $V_0 = \{a, b, c\}$ and $E_0 = \{(a, c), (b, c)\} \cup \{(a, b)\}$ and let $G'_1 = (V'_1, E'_1)$ be the root of the s-tree S'_1. By Definition 3.6, G_0 and G'_1 are the modified split graphs of G with respect to the separation pair $\{a, b\}$. Since G'_1 is s-tree decomposable, by induction hypothesis it is tree decomposable. Therefore there is a tree decomposition T'_1 of G'_1.

Build a tree decomposition T such that its subtrees are T'_1, $\{a, c\}$ and $\{b, c\}$. See Table 1. Subtrees $\{a, c\}$ and $\{b, c\}$ share the node c. Since $\{a, b\}$ is a separation pair of G, $V_0 \cap V'_1 = \{a, b\}$. But $c \in V_0$, thus $c \notin V'_1$, $\{a, c\} \cap V'_1 = \{a\}$, and $\{b, c\} \cap V'_1 = \{b\}$. Therefore T is a tree decomposition of G.

Applying the same procedure for cases in rows three and four in Table 1 completes the proof. □

5.2 Domain equivalence

Now, we will see that the class of s-tree decomposable geometric constraint graphs and the class of tree decomposable graphs are the same. In other words, a geometric constraint problem expressed by means of a geometric constraint graph is solvable by Owen's technique if and only if the graph is tree decomposable. Since we proved in [9] that a geometric constraint graph is solvable by reduction analysis, [5], if and only if the graph is tree decomposable, this implies that Owen's technique and reduction analysis have the same domain and that its domain can be characterized by the class of tree decomposable graphs.

Theorem 5.4 *Let $G = (V, E)$ be a geometric constraint graph. The following assertions are equivalent:*

1. *G is tree decomposable.*
2. *G is s-tree decomposable.*
3. *G is solvable by reduction analysis.*
4. *G is solvable by decomposition analysis.*

Proof
Joan-Arinyo *et al.* proved in [9] the equivalence of assertions 1, 3 and 4. Lemma 5.2 proves that 1 implies 2 and Lemma 5.3 proves that 2 implies 1. □

6 Summary

We have introduce the concept of *deficit* of a constraint graph. Based on this concept, we have presented a new formalization for a decomposition analysis algorithm. We have proved its correctness. The idea of preserving just the constraint graph deficit avoids the need for general algorithms to compute triconnected components. Thus the resulting decomposition analysis algorithm is simpler both conceptually and from a computational point of view.

We have used the *tree decomposition* as a general tool for decomposition analysis of constraint graphs. Specifically, we have applied it to prove that different decomposition analysis formalizations solve the same class of geometric constraint problems.

Acknowledgements

This research has been supported by CICYT under the project TIC2001-2099-C03-01.

References

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Computer Science and Information Processing. Addison Wesley Publishing Company, Reading, MA, 1974.

[2] Gary Chartrand and Linda Lesniak. *Graphs & Digraphs*. Chapman & Hall, 3rd edition, 1996.

[3] C. Durand. *Symbolic and Numerical Techniques for Constraint Solving*. PhD thesis, Purdue University, Department of Computer Sciences, December 1998.

[4] I. Fudos. Editable representations for 2D geometric design. Master's thesis, Purdue University, Department of Computer Sciences, 1993.

[5] I. Fudos and C.M. Hoffmann. Correctness proof of a geometric constraint solver. Technical Report CSD 93-076, Department of Computer Sciences, Purdue University, December 1993.

[6] I. Fudos and C.M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, 16(2):179–216, April 1997.

[7] C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Geometric constraint decomposition. In B. Brüderlin and D. Roller, editors, *Geometric Constraint Solving and Applications*, pages 171–195. Springer, Berlin, 1998.

[8] J. E. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. Technical report, Computer Science Department. Cornell University, Ithaca, NY. USA, February 1974. New revision of TR 72-140.

[9] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana. On the domain of constructive geometric constraint solving techniques. In R. Ďuricovič and S. Czanner, editors, *Spring Conference on Computer Graphics*, pages 49–54, Budmerice, Slovakia, April 25-28 2001. IEEE Computer Society, Los Alamintos, CA.

[10] G. Laman. On graphs and rigidity of plane skeletal structures. *Journal of Engineering Mathematics*, 4(4):331–340, October 1970.

[11] N. Mata. Solving incidence and tangency constraints in 2D. Technical Report LSI-97-3R, Department LSI, Universitat Politècnica de Catalunya, 1997.

[12] Gary L. Miller and Vijaya Ramachandran. A new graph triconnectivity algorithm and its parallelization. *Combinatorica*, 12:53–76, 1992.

[13] J.C. Owen. Algebraic solution for geometry from dimensional constraints. In *ACM Symp Foundations of Solid Modeling*, pages 397–407, Austin, TX, 1991.

[14] P. Todd. A k-tree generalization that characterizes consistency of dimensioned engineering drawings. *SIAM J. Disc. Math*, 2(2):255–261, 1989.