Treedecomposition of Geometric Constraint Graphs Based on Computing Graph Circuits

Robert Joan Arinyo, Marta Tarrés Puertas, Sebastià Vila Marta
Grup d'Informàtica a l'Enginyeria Universitat Politècnica de Catalunya Barcelona, Catalonia

ABSTRACT
The graph-based geometric constraint solving technique works in two steps. First the geometric problem is translated into a graph whose vertices represent the set of geometric elements and whose edges are the constraints. Then the constraint problem is solved by decomposing the graph into a collection of subgraphs each representing a standard problem which is solved by a dedicated equational solver.

In this work we report on an algorithm to decompose biconnectedtree-decomposable graphs representing either underor wellconstrained 2D geometric constraint problems. The algorithm recursively first computes a set of fundamental circuits in the graph then splits the graph into a set of subgraphs each sharing exactly three vertices with the fundamental circuit. Practical experiments show that the reported algorithm clearly outperforms the treedecomposition approach based on identifying subgraphs by applying specific decomposition rules.

1. INTRODUCTION
Geometric models are data structures designed to represent and describe objects. The main properties encoded include geometric shape and topological properties.

Computer Aided Design (CAD) systems are software applications built to help industrial designers in the product design cycle. Because the main activity of CAD systems is related to manage descriptions of objects, geometric models are at the core of such systems. Parametric CAD systems partially capture the design intent by defining functional relationships between dimensional variables and geometric elements, that is, by applying constraint-based geometric design.

A paramount issue found in constraint-based geometric design is the constraint solving problem which can be roughly summarized as follows: Given a set of geometric elements and a set of constraints between them, place each geometric element in such a way that the constraints are fulfilled. The algorithms that solve geometric constraint problems are named solvers. The reader is referred to the work in [2, 9, 18], for an extensive review on geometric constraint solving algorithms.

Among the existing solving methods we focus on constructive techniques. In these techniques the input is a geometric constraint problem represented as a geometric constraint graph. Then the graph is decomposed yielding as output a constructive plan, that is, a sequence of basic steps that describe how to build a solution to the constraint-based geometric problem. Basic steps correspond to elemental operations which are solved with dedicated algorithms.

When decomposing geometric constraint graphs two different cases can be considered according to the graph connectivity. For 0 and 1 connected graphs, there is a smooth approach. We refer to [12] for the details. For biconnected graphs, the approach is far more difficult. In what follows we focus on this class of graphs.

In this work we introduce a new algorithm to decompose biconnected tree-decomposable graphs representing either under- or wellconstrained 2D geometric constraint problems. The algorithm is based on the decomposition of a graph in the set of bridges induced by a fundamental circuit. It is inspired by the work of Miller and Ramachandran, [14], developed with the aim of finding the set of triconnected components of a given graph. The approach does not need a set of rules defined a priori which identify subgraphs with specific configurations. It figures out the decomposition by a direct computation First the algorithm computes a set of fundamental circuits in the graph. Then a fundamental circuit is conveniently selected and the graph is split into a set of subgraphs each sharing exactly three vertices with the selected fundamental circuit, following the tree decomposition technique reported in [13].

The rest of the paper is organized as follows. In Section 2 we review some previous work related to graph-based, constructive geometric constraint solving methods. In Section 3 we recall basic terminology of graph theory, define the basic geometric constraint solving we are dealing with, and the tree decomposition of a constraint graph. Section 4 is devoted to the new decomposition algorithm. Section 5 reports on the runtime resulting from empirically testing the algorithm. Finally, in Section 6 we offer a brief summary and outline future work.

2. RELATED WORK

Many attempts to provide general, powerful and efficient constructive graph-based techniques to solve geometric constraint problems have been reported in the literature. For an extensive review see the works in [5, 7, 8, 11, 16]. These techniques decompose the geometric constraint graph into a set of basic subgraphs where each of them represents a standard problem which can be solved by a fixed algorithm or an equational solver.

Hoffmann et al. [7, 8] described a flow-based method for decomposing the graph of a geometric constraint problem based on degrees of freedom calculations. The method first introduces dense graphs which are considered the basic graphs that will not be further split. The algorithm decomposes a given constraint graph into a set of dense subgraphs using a network flow algorithm. This approach is general however operations needed to solve dense graphs can be arbitrarily complex and not necessarily have geometric meaning.

A family of graph-based algorithms carry out the constraint graph decomposition by applying a set of predefined rules which preserve geometric meaning. These algorithms have a limited scope but are efficient and solve a substantial set of geometric constraint problems arising in constraint-based parametric modeling. Owen in[16] described a top-down algorithm for computing a decomposition of an arbitrary constraint graph. The algorithm recursively splits the graph into split components. 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 basic subgraphs. The algorithm closely follows the triconnected components decomposition of [10]. In [13], it is proved that the worst case running time complexity of this algorithm is $O(n^2)$.

Fudos and Hoffmann, [5], reported on two graph-based constructive approaches to solve systems of geometric constraints. The top-down method is roughly equivalent to the method by Owen, [16]. The bottom-up method, named reduction analysis, begins by computing a set S of basic subgraphs. Then graphs in S are iteratively merged until a unique graph which contains all the geometric elements in the problem is obtained. Fudos and Hoffmann claim the algorithm to have a $O(n^2)$ runtime complexity in the worst case.

Gao et al. in [6], report on a method that from the constraint graph extracts a binary connectivity tree, called C-tree, according to a well defined set of rules. For each node in the C-tree, the left child represents a subproblem which can be solved. Then the subtree rooted in the right child is recursively visited and merged with the rigid body coming from the left child. The method is essentially equivalent to those described above.

The tree decomposition of a constraint graph was introduced by Joan-Arinyo et al. in [13]. We shall review this concept in Section 3. Tree decompositions have been also useful from a theoretical point of view. Moreover, they are also a suitable representation for constructive plans. Tree decomposition is the technique underlying the graph-based geometric constraint solving framework SolBCN, [19], that has been used to develop the technique reported here.

In these approaches, the graph decomposition is carried out by splitting it into three subsets of vertices that pairwise share just one element. These shared vertices are called hinges. In general, hinges are identified by an almost exhaustive search therefore it is a time consuming computation. Thus devising efficient algorithms based on direct computation of hinges would be a great accomplishment.

3. PRELIMINARIES

In this section we recall basic terminology of graph theory, define the basic geometric constraint solving we are dealing with, and the tree decomposition of a constrint graph. For more information we refer the reader to the works by Hoffmann et al., [9], Joan-Arinyo et al., [13, 12], Whitney, [21], Even, [3], and Thulasiraman and Swamy, [20].

3.1 Graph Concepts

A graph G = (V,E) consists of a set of vertices V , also called nodes, and a set of edges E. An edge $e \in E$ is a pair of vertices $e = (v_i, v_j)$ such that $v_i, v_j \in V$. Vertices v_i and v_j associated with and edge e are called the end vertices of e. In general, V (G) will denote the set of vertices and E(G) the set of edges of a graph G. A graph can be represented by a diagram in which a vertex is symbolized by a dot and an edge by a line segment connecting two dots.

The number of edges incident on a vertex v is called the degree of the vertex, and is denoted by d(v).

A walk in a graph G = (V,E) is a finite alternating sequence of vertices and edges $[v_0, e_1, v_1, e_2, \ldots, v_{n-1}, e_n, v_n]$ beginning and ending with vertices such that v_{i-1} and v_i are the vertices bounding edge e_i, $1 \leq i \leq n$. A walk is a trail if all its edges are distinct. Note that any trail is itself a graph. A trail is open if its end vertices are distinct. An open trail is a path if all its vertices are distinct. A closed trail is a circuit if all its vertices except the end vertices are distinct. Clearly, vertices in a circuit have a cyclic order, that is, circuits $C = \{c_1, c_2, \ldots, c_{n-1}, c_n\}$ and $C' = \{c_n, c_1, c_2, \ldots, c_{n-1}\}$ are the same circuit. Thus, once a circuit has been defined and a given vertex selected as the first in the cycle, we can consider that vertices are always labeled as $C = \{c_1, c_2, \ldots, c_{n-1}, c_n\}$ without loss of generality.

A graph G = (V,E) is connected if there exists a path between every pair of vertices in G, otherwise G is disconnected. The maximal connected subgraphs of a disconnected graph G are the connected components of G.

Let G = (V,E) be a connected graph. According to [20], we say that a vertex $v \in V$ is an articulation vertex if the subgraph induced in G by $\{V(G) - v\}$ is disconnected. A vertex $v \in V$ is an articulation vertex if and only if there are vertices $u,w \in V$, with $u \neq v$ and $w \neq v$ such that v is on every $u - w$ path.

A non-separable or biconnected graph G = (V,E) has no articulation vertices, otherwise it is separable. See [21].

A biconnected component of a connected graph G is a maximal biconnected subgraph of G. A connected graph can be decomposed into biconnected components. For any biconnected graph G = (V,E), given a pair of vertices $u, v \in V$ with $u \neq v$, there are, at least, two disjoint paths $u - v$.

The connectivity of a graph G is the minimum number k of vertices that must be removed to disconnect G. If the connectivity of G is k, we write $\kappa(G) = k$. For a disconnected graph G, $\kappa(G) = 0$. For a connected graph G, we have $\kappa(G) \geq 1$. A separable graph G has $\kappa(G) = 1$. For a biconnected graph G has $\kappa(G) \geq 2$. A graph G with $\kappa(G) \geq 3$ is called triconnected. Biconnected graphs can be decomposed into triconnected components.

A graph is said to be acyclic if it has no circuits. A tree of a graph G is a connected acyclic subgraph of G. A spanning tree T for a graph G is a tree that connects all the vertices in V . The edges of a spanning tree T are called the branches of T. The edges of G that are not in T are called the chords.

Let G = (V,E) be a graph with |V | = n and let G' be a graph such that $G' \subset G$. Following [20], G' is said to be a spanning tree of G if and only if G' is acyclic, connected, and has $n - 1$ edges.

Let G = (V,E) be a graph with |V | = n and |E| = m. Let T be a spanning tree to G with $b_1, \ldots, b_{n-1}$ branches and $c_1, \ldots, c_{m-n+1}$ chords of T. The graph resulting from adding to T one chord, say c_i, contains exactly one circuit C which consists of the chord c_i and those branches of T that lie in the unique path in T between the end vertices of c_i. The circuit C is a fundamental circuit of G with respect to the chord c_i of the spanning tree T.

The $m-n+1$ possible fundamental circuits $C_1, \ldots, C_{m-n+1}$ of G with respect to the chords of the spanning tree T of G is known as a set of fundamental circuits.

Consider a graph G = (V,E). We say that G is embeddable in a surface Σ if G can be drawn in Σ in such a way that: (1) each vertex $v \in V$ is represented by a point in Σ, (2) each edge $e \in E$ is represented by a continuous curve $c \in \Sigma$ connecting the two points which represent its end vertices, and (3) no two curves

share any point but the vertices. Such a drawing is called an embedding of G in Σ. A graph G embedded in the Euclidean plane is said to be planar.

Let G be a graph, Σ the Euclidean plane and D an embedding of G in Σ. A face F of D is a maximal region of Σ bounded by edges of D such that for any pair of points (x, y) in F, there is a continuous curve c that connects x to y with c ∈ F

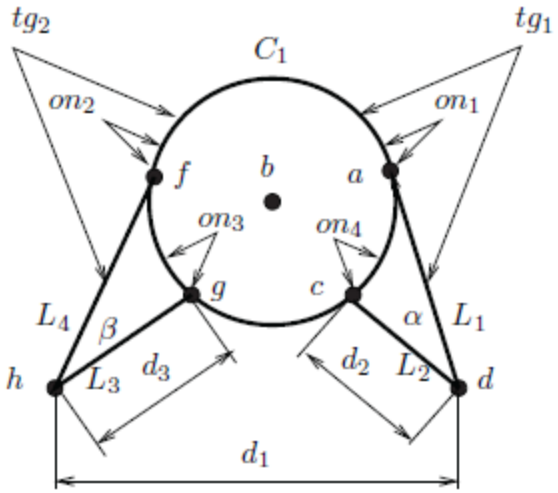

Figure 1: Case study. Geometric problem defined by constraints.

3.2 The Basic Constraint Problem

In this paper we focus on *the basic constraint problem* defined as follows. Given a set of geometric elements and a set of constraints between them, place each geometric element in such a way that the constraints are fulfilled. We consider 2D geometric constraint problems defined by a set of geometric elements like points, lines, line segments, circles and circular arcs with fixed radius, along with a set of constraints like distance, angle, incidence and tangency between any two geometric elements. Figure 1 shows an example of a geometric problem defined by constraints. The problem includes seven points, {a, b, c, d, f, g, h}, four straight segments, {L_1, L_2, L_3, L_4}, one fixed-radius circle, {C_1}, and eleven constraints: point-point distance {d_1, d_2, d_3}, angle betweeen two straight segments {α, β}, segment-circle tangency {tg_1, tg_2} and point-on {on_1, on_2, on_3, on_4}.

The geometric constraint problem can be represented by means of a geometric constraint graph G = (V,E), where the
nodes in V are geometric elements with two degrees of freedom and the edges in E are geometric constraints such that each of them cancels one degree of freedom. Figure 2 shows the graph G = (V,E) derived from the geometric constraint problem given in Figure 1 where the set of vertices is V = {a, b, c, d, f, g, h} and the set of edges is E = {(a, d), (a, c), (a, b), (b, c), (b, g), (b, f), (f, g), (f, h), (g,h), (h, d), (d, c)}.

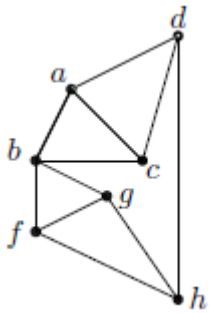

Figure 2: Case study. Constraint graph.

Once a geometric constraint problem has been translated into a geometric constraint graph, solving the geometric constraint problem amounts to decompose the graph until basic configurations are found to

which standard equational solvers are applied. Therefore, devising feasible algorithms that efficiently decompose constraint graphs is paramount.

3.3 Tree Decompositions

Following Fudos and Hoffmann, [5], Joan-Arinyo et al. defined in [12] the concept of set decomposition as a way of partitioning a given abstract set. Let S be a set with at least three different members, say a, b, c. Let $S_1, S_2, S_3 \subset S$. We say that $\{S_1, S_2, S_3\}$ is a set decomposition of S if $S_1 \cup S_2 \cup S_3 = S$ and $S_1 \cap S_2 = \{a\}$ and $S_2 \cap S_3 = \{b\}$ and $S_1 \cap S_3 = \{c\}$. Vertices a, b, c are called the hinges of the set decomposition, and S_1, S_2 and S_3 are the clusters, [5]. Notice that a set decomposition is not necessarily unique.

The set decomposition of a graph is defined as follows. Let G = (V,E) be a graph and let V_1, V_2 and V_3 be 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.

Finally, we define the concept of tree decomposition of a graph. 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 node $V' \subset V$ 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.

Geometric constraint graphs for which there is a tree decomposition are known as tree decomposables, that is, the associated geometric constraint problem is solvable by the tree decomposition approach. It is said that the underlying constraint problem is quadratically solvable becasue it can be solved after translating it into a set of degree two equations.

4. THE ALGORITHM

Let G = (V,E) be a geometric constraint graph. Given a set of hinges $\{v_1, v_2, v_3\} \subseteq V$, a set decomposition of G can be trivially computed. Moreover, a recursive application of set decompositions yields a tree decomposition of G. Therefore, the problem of computing a tree decomposition of G can be reduced to computing sets of hinges.

4.1 Algorithm Overview

The algorithm, outlined in Figure 3, computes one decomposition step by splitting a graph into three connected components. It is based on the following result (the proof is omitted for the sake of conciseness)

Theorem Let D be a planar embedding of a biconnected graph G = (V,E). Then, vertices $\{v_1, v_2, v_3\} \subset V$ are hinges of V if and only if there are two faces F_1 and F_2 in D whose bounding edges have vertices $V(F_1)$, $V(F_2)$ such that $\{v_1, v_2, v_3\} \subseteq V(F_1) \cap V(F_2)$.

The algorithm proceeds as follows. Let G = (V,E) be a biconnected geometric constraint graph derived from a geometric constraint problem. First a spanning tree for the graph G is computed by applying a depth-first search. Then the associated fundamental circuits $\{C_1, \ldots, C_n\}$ are identified.

INPUT: biconnected constraint graph G = (V,E), $|V| \geq 3$
OUTPUT: a set of hinges $\{v_1, v_2, v_3\} \subseteq V$, if one exists

Compute a spanning tree T of G
Compute the set of fundamental circuits C of G according to T
foreach $C_i \in C$ **do**
    Compute the set of bridges B of G with respect to C_i
    Compute the collapsed graph G'
    Compute the merged graph G''
    Compute the planar embedding D of G''
    foreach $F \in D$ **do**
        foreach $\{v_1, v_2, v_3\} \subseteq F$ **do**
            if $\{v_1, v_2, v_3\} \in C_i$ **then**
                return $\{v_1, v_2, v_3\}$
            endif
        endfor
endfor

endfor
return ∅

Figure 3: Decomposition of a biconnected graph.

According to [12], any set of hinges of G must be a subset of the vertices of some fundamental circuit C_i of G. Therefore we fix one of the faces, F_1, F_2, by restricting the search for hinges to faces bounded by fundamental circuits. The search is performed in a planar embedding D of a graph G' resulting from transforming the given graph G according to the bridges (to be defined later on) induced in G by the fundamental circuit under study. If the algorithm fails in finding a fundamental circuit with a set of hinges, the input graph cannot be decomposed by our method. In what follows we detail each step in the algorithm.

4.2 Computing Fundamental Circuits

To illustrate the ideas, in what follows we will refer to the case study graph G = (V,E) depicted in Figure 2.

To identify a set of fundamental circuits, first a spanning tree T to the constraint graph G is computed using a depth first spanning tree algorithm, [20]. Assume that vertex d is chosen as the initial node for T. Then Figure 4a shows the resulting depth first spanning tree of G where edges in T are in bold lines and chords are drawn as dotted lines. Since each chord induces one fundamental circuit, there are five fundamentals circuits, shown in Figure 4b. The algorithm to connect each circuit is trivial and we do not detail it here.

4.3 Computing Bridges

Given a constraint graph G = (V,E) and one circuit C ⊆ G, our algorithm computes the set of bridges induced in G by the circuit C, [3, 12].

First we define the concepts of non-singular and singular components of a non-separable graph induced by a subset of vertices. Let G = (V,E) be a non-separable graph and let S ⊆ V . Consider the partition of the set V − S into classes

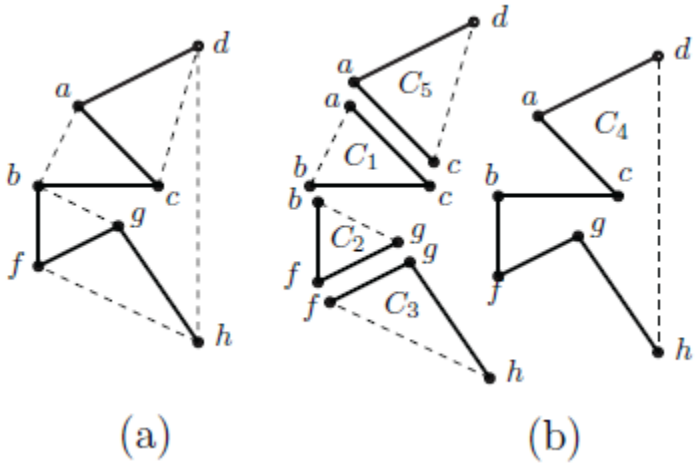

Figure 4: Case study. a) Spanning tree. b) Fundamental circuits C_1, C_2, C_3, C_4 and C_5.

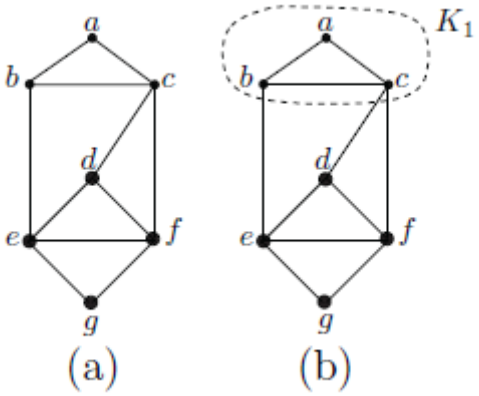

Figure 5: a) Graph. b) Classes.

such that two vertices are in the same class if and only if there is a path connecting them which does not include any vertex of S. Let ≡ denote this relationship and assume that $V - S/_{\equiv} = \{K_1, \ldots, K_m\}$. With each class K_i we associate a subgraph $H = (V', E') \subseteq G$ defined as

1. $V' \supset K_i$

2. V' includes all the vertices of S which are connected by an edge to a vertex of K_i in G

3. $E' \subseteq E$ contains all edges of G which have at least one end vertex in K_i

The subgraph $H = (V', E')$ is a non-singular component of G induced by S.

To illustrate the concept of non-singular component, consider the graph G = (V,E) in Figure 5a and let S = {d, e, g, f}. Then V − S = {a, b, c} and there is just one induced class K_1 = {a, b, c} shown in Figure 5b enclosed in a dotted line. The associated non-singular component is the subgraph with vertices V' = {a, b, c, e, d, f} and edges E' = {(a, b), (a, c), (b, c), (b, e), (c, d), (c, f)}, denoted as H_1 in Figure 6a.

With each edge (u, v) such that u, v ∈ S we associate a subgraph in G, called singular component, with vertices V' = {u, v} and edges E' = {(u, v)}. For example, with edge (e, f) in Figure 5 we associate the singular component denoted H_6 in Figure 6a. Similarly, with each edge in S, we associate a singular component. Notice that a set of vertices S induces a graph decomposition into a set of singular and non-singular components such that two components share no edges, and the only vertices they can share are vertices of S.

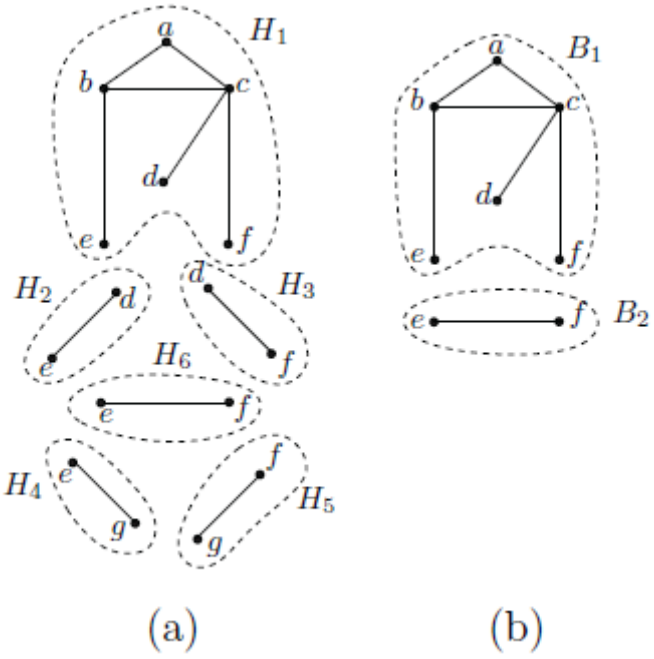

Figure 6: a) Components. b) Bridges.

Now we define the concept of bridge. Let H be the set of singular and non-singular components induced by S □ V in the graph G = (V,E). A bridge, B_i, is a component $H_i \in H$ such that the vertices of its edges are not contained in S. A bridge is said to be singular if the corresponding component is singular otherwise it is non-singular. The attachment of a component H_i is the set of vertices shared with S, $at(H_i) = V(H_i) \cap S$. Figure 6b shows the bridges B_1 and B_2 corresponding to components H_1 and H_6. B_1 is a non-singular bridge while B_2 is singular. The attachments are respectively $at(B_1)$ = {e, d, f} and $at(B_2)$ = {e, f}.

In the case study shown in Figure 2, assume that S is the fundamental circuit C_4 = [d, a, c, b, f, g, h] shown in Figure 4b. The components induced in G by C_4 are

$H_1 = (\{a, b\}, \{(a, b)\})$
$H_2 = (\{c, d\}, \{(c, d)\})$
$H_3 = (\{b, g\}, \{(b, g)\})$
$H_4 = (\{f, h\}, \{(f, h)\})$

Notice that H_i, $1 \leq i \leq 4$ are also the bridges with respect to C_4. Moreover, in this case, all the bridges are singular. Figure 7 outlines the algorithm that computes the set of bridges.

4.4 Computing the Collapsed Graph

Once a set of bridges has been identified, we simplify the given graph. We start by introducing some required concepts.

A star graph is a connected graph G = (V,E) with one distinguished vertex x ∈ V , called center, the degree of which

```
INPUT: biconnected graph G = (V,E), |V | ≥ 3, C a fundamental circuit of G
OUTPUT: The set of bridges B induced by C in G
/* Compute the singular bridges */
foreach a, b ∈ V (C) do
        if (a, b) ∈ E and (a, b) ‘∈ C then
                B := B ∪ ({a, b}, {(a, b)})
        endif
endfor
/* Compute the non-singular bridges */
N = V − V (C)
Let G’ ⊆ G be the graph induced by N ⊆ V
foreach G’’ ∈ ConnectedComponents(G’) do
        foreach (v,w) such that v ∈ V (G’’),w ∈ V (C) do
          if (v,w) ∈ E(G) then
            Add (v,w) to G’’
          endif
        endfor
        Add G’’ to B
endfor
```

Figure 7: Computing the set of bridges.

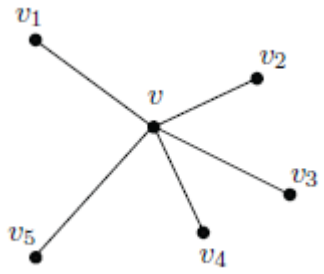

Figure 8: Star graph example.

is d(x) > 1 and such that for each $v_i \in V$ with $v_i \neq x$, $d(v_i) = 1$. Figure 8 depicts a star graph example.

Let $B = (V_B,E_B)$ be a bridge. We define the star graph of B, noted s(B), as the graph with vertices $\{x\} \cup at(B)$ and edges $\{(x, v) \mid v \in at(B)\}$ where x is a new vertex. We will refer to s(B) as a star bridge.

Let G = (V,E) be a graph and C a fundamental circuit of G. Let $B = \{B_1, \ldots, B_n\}$ be the set of bridges induced by C in G. The collapsed graph of G is the graph resulting from replacing each bridge $B_i \in B$ with the corresponding star bridge $s(B_i)$.

Considering that in the case study depicted in Figure 4a, the bridges induced in G by the circuit C_4 are H_1,H_2,H_3,H_4, the collapsed graph resulting by replacing each bridge H_i with its star bridge $S_i = s(H_i)$ is shown in Figure 9a.

4.5 Computing the Merged Graph

We will further simplify the collapsed graph by merging sets of star bridges into larger star bridges. The resulting graph will be called the merged graph.

Let $B = \{B_1, \ldots, B_n\}$ be the set of star bridges in a collapsed graph. Let B_i and B_j be two different star bridges in B. We define the merging of B_i and B_j as the star bridge $B_{ij} = (V_B,E_B)$ with $V_B = \{x\} \cup at(B_i) \cup at(B_j)$ and $E_B = \{(x, v) \mid v \in at(B_i) \cup at(B_j)\}$ where the center x is a new vertex.

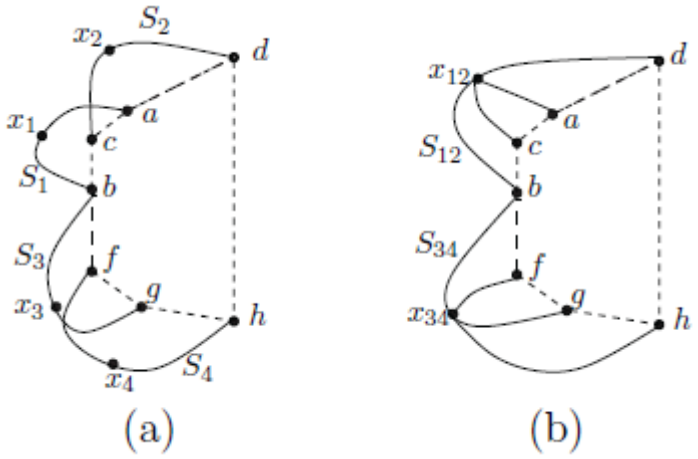

Figure 9: a) Collapsed graph. b) Merged graph.

Let B be the set of bridges induced in a graph G by the circuit C and let B_i,$B_j \in B$. We say that B_i and B_j interlace if one of the two following conditions hold:

1. There are four different vertices a, b, c, d ∈ C with a, b ∈ at(B_i) and c, d ∈ at(B_j) such that they are placed in C in the sequence [a, c, b, d], or
2. The attachments at(B_i) and at(B_j) share three vertices.

Now computing the merged graph is straightforward. Choose two different interlacing star bridges, say B_i and B_j, and replace them with their star merge B_{ij} . Repeat this process until no two stars interlace. Since no two star bridges interlace in a merged graph, the merged graph always has a planar embedding. See Even, [3].

Figure 9b shows a merged graph corresponding to the collapsed graph in Figure 9a. Notice that star bridges S_1 and S_2 have been merged into S_{12}, and star bridges S_3 and S_4 have been merged into S_{34}. Clearly the merged graph is a planar graph.

4.6 Computing the Planar Embedding

Algorithms to compute planar embeddigns of planar graphs trace back to as far as F´ary [4]. More recent developments are collected by Nishizeki and Chiba in [15] and by Skiena [17]. However, having in mind the properties of our merged planar graph and the specifity of the embedding we are looking for, we have developed an ad hoc simple algorithm.

Although we are not interested in actually drawing a picture of our constraint graphs, recalling some standard basic properties of planar graphs drawings will help in our rational. A planar graph can be drawn in the two-dimensional space with no two of its edges crossing. Such a drawing of a planar graph is called a plane drawing.

Any plane drawing separates the plane into distinct regions bordered by graph edges called faces. As a simple example, any embedding of a circuit C into the plane separates it into two faces: the region inside the circuit, say F_b, and the (unbounded) region outside the circuit, say F_u. The unbounded region outside the graph's embedding is called the outer face. See Figure 10.

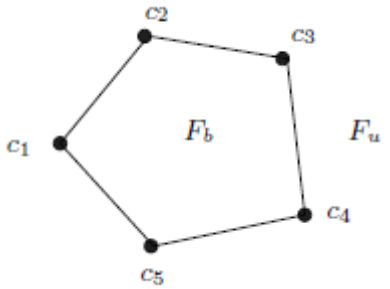

Figure 10: A circuit embedded in the plane defines two faces F_b and F_u.

Let C be the circuit considered in the constraint graph and G be the induced merged graph. We start the embedding of G by trivially embedding C. Before proceding to include the bridges of G in the initial embedding, we need to introduce a few more concepts.

Since a circuit and the star bridges it induces in a graph share the vertices in the bridges attachments, we will consider that vertices in a bridge attachment are sorted accordingly to the vertices in the circuit. If $C = \{c_1, c_2, \ldots, c_n\}$ is the circuit under consideration, in what follows the closed interval $[c_1, c_3]$, called segment, will denote the subset of vertices $\{c_1, c_2, c_3\}$ while the open interval (c_1, c_3) will denote the subset of vertices $\{c_2\}$.

The span of a bridge B, noted sp(B), is the open interval over V (C) whose lower and upper bounds are respectively the first and last vertices in the attachment at(B).

The set of segments defined by B, noted se(B), is the set of segments over V (C) defined by two consecutive vertices in the attachment at(B).

Figure 11 shows a circuit with vertices $\{c_1, c_2, \ldots, c_{10}\}$ and a bridge B with attachment $at(B) = \{c_1, c_7, c_9\}$. The span of B is $sp(B) = (c_1, c_9) = \{c_2, c_3, c_4, c_5, c_6, c_7, c_8\}$ and the set of segments defined by B is $se(B) = \{[c_1, c_7], [c_7, c_9]\}$.

Let F be a face in a given planar embedding of a graph with respect to a given circuit C. We say that the characteristic vertices of F is the set $ch(F) = \{v \mid v \in V(F) \cap C\}$. Note that ch(F) is always a segment of C.

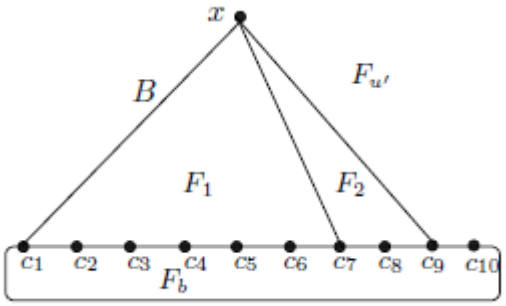

Figure 11: A bridge B attached to a circuit.

INPUT:A circuit C and a set of merged bridges $B = \{B_1, \ldots, B_n\}$
OUTPUT: a planar embedding D

Sort the set of bridges B
$D = \{F_u\}$
for B_i in B **do**
    Select $F \in D$ such that $at(B_i) \subseteq ch(F)$
    Remove F from D
    Add to D the face characterized by $ch(F) - sp(B)$
    for I in se(B) **do**
        Add to D the face characterized by $ch(F) \cap I$
    endfor
endfor
return D

Figure 12: Computing the planar embedding of the merged graph.

The embedding is computed as a list of faces each represented by a circuit that defines the face border. Assume that the embeding of the considered circuit C is $D = \{F_u\}$, illustrated in Figure 10. Then star bridges $B_i \in B$ are iteratively included in the current embedding following the algorithm outlined in Figure 12. We assume, without loss of generality, that each new star bridge is embedded (it is drawn) in a face different from the one labeled F_b, thus face F_b is never split and we will ignore it in what follows.

First the set of bridges is sorted according to decreasing number of vertices in the spans. When two bridges have the same number of vertices in their spans, they are sorted according to increasing number of vertices in the attachments. Then, for each bridge B_i to be added to the current embedding, we identify the face where it should be embedded. Clearly, if $D = \{F_1, \ldots, F_m, F_u\}$ is the current embedding, B_i should be embedded in a face, say F_k, such that $at(B_i) \subseteq ch(F_k)$. Since bridges do not interlace, two merged bridges at most share vertices that are bounds of bridges spans and faces are just embeddings of bridges, face F_k is unique. Then face F_k is replaced with the set of faces that B_i induces in it, defined as follows

1. One face F_{k1} whose border is defined by replacing in the border of F_k the set of vertices $sp(B_i) \cap ch(F_k)$ with a new vertex x_{k1}.
2. For each segment in $se(B_i) = \{S_1, \ldots, S_j, \ldots, S_n\}$ a new face $F_{k(j+1)}$ is included the border of which is defined by appending to the segment S_j the vertex x_{k1}.

To illustrate how a new bridge is added, assume that the current embedding is the one depicted in Figure 11 and that

the next merged bridge to be included is B_i given in Figure 13. Since at(B_i) = {c_1, c_3, c_4, c_6} ⊂ ch(F_1), the face to be replaced is F_1. The set of new faces includes F'_1 plus F_3, F_4 and F_5, shown in Figure 14. The tree of segments is

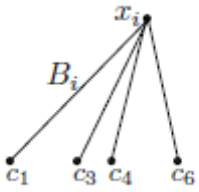

Figure 13: A bridge to be included in the embedding in Figure 11.

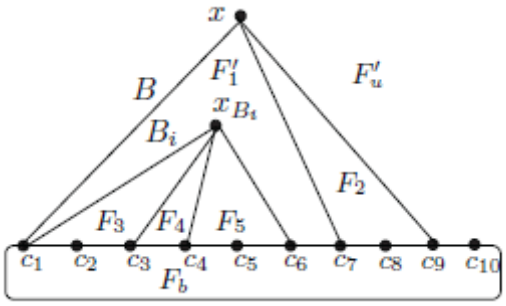

Figure 14: Embedding after including a new merged bridge B_i.

given in Figure 15 and the list of faces is

F'_u = [c_1, x, c_9, c_{10}] F_3 = [c_1, c_2, c_3, x_{B_i}]
F_1 = [c_1, x_{B_i}, c_6, c_7, x] F_4 = [c_3, c_4, x_{B_i}]
F_2 = [c_7, c_8, c_9, x] F_5 = [c_4, c_5, c_6, x_{B_i}]

Considering the case study, the embedding for the merged graph shown in Figure 9b is given in Figure 16 where the resulting embedding includes the faces {F_u, F_1, F_2, F_3, F_4, F_5, F_6}. The borders of the faces in the planar embedding are the circuits:

F_u = [d, x_{12}, b, x_{34}, h] F_4 = [b, x_{34}, f]
F_1 = [d, x_{12}, a] F_5 = [f, x_{34}, g]
F_2 = [a, x_{12}, c] F_6 = [g,x_{34}, h]
F_3 = [c, x_{12}, b]

The embedding is computed while building a linear range search tree, [1], to efficiently identify the face in the current embedding which must be replaced when a new bridge is embedded. Each tree node stores a segment of vertices and a pointer to the face it defines, stored in the list of faces. The root node stores the set of vertices of face F_u or, equivalently, the set of verices in the circuit under consideration.

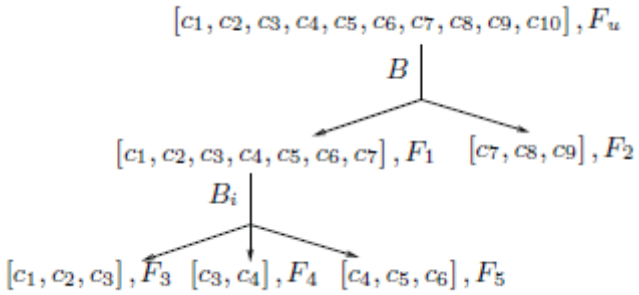

Figure 15: Tree of segments for the embedding in Figure 14.

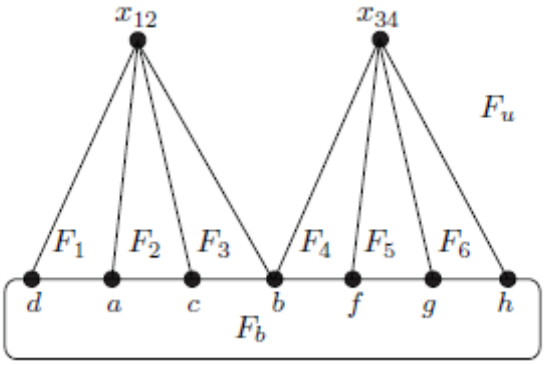

Figure 16: Planar embedding for the merged graph given in Figure 9b.

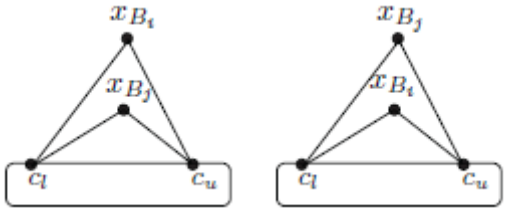

Figure 17: Embeding two bridges with two vertices in their attachments and the same span.

To show that the sorting defined above guarantees that new bridges are added to the tree always as tree leaves, we need to consider four different cases. First, let B_i,B_j be two bridges such that $|sp(B_i)| > |sp(B_j)|$. Since bridges do not interlace, faces induced by B_i and B_j either are disjoint or the faces induced by B_j lie inside some face induced by B_i. Now consider the situation where $|sp(B_i)| = |sp(B_j)|$. First assume that $|at(B_i)| \geq 3$ and $|at(B_j)| \geq 3$. Since bridges do not interlace, faces induced by B_i and B_j must be disjoint. Therefore order does not matter. Second let $|at(B_i)| \geq 3$ and $|at(B_j)| = 2$. Since bridges do not interlace, faces induced by B_i and B_j either are disjoint and the order does not matter or $sp(B_i) \subseteq sp(B_j)$ and the face coming from B_j is included in the tree before faces induced by B_i. Finally let $|at(B_i)| = |at(B_j)| = 2$. If B_i and B_j are disjoint the ordering does not matter. If $sp(B_i)$ and $sp(B_j)$ are coincident the set of faces in the resulting embedding depends on the embedding sequence, see Figure 17. However note that the set of intervals generated in the circuit is coincident. Therefore the embedding order does not affect the set of hinges induced in the circuit.

4.7 Computing Hinges

Finally we have to search for hinges in the planar embedding.

Let G = (V,E) be the constraint graph, C a circuit of G and D the planar embedding for the merged graph. We say that three vertices in C, say {a, b, c}, are hinges if there is a face F in D with $F \neq F_b$ such that $\{a, b, c\} \in ch(F)$.

Therefore, to compute a set of hinges the algorithm searches for a face $F \in D$ such that $|ch(F)| \geq 3$. Any subset of three vertices of F is a set of hinges.

Consider the embedding shown in Figure 16. The set of faces is $\{F_1, F_2, F_3, F_4, F_5, F_6, F_u\}$. Since {d, b, h} is a subset of $ch(F_u)$, {d, b, h} is a set of hinges. Therefore the initial graph given in Figure 2 can be tree decomposed into three clusters G_1, G_2 and G_3 as shown in Figure 18.

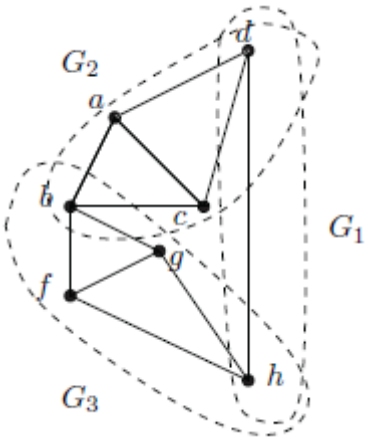

Figure 18: Graph and clusters G_1, G_2 and G_3.

5. EXPERIMENTAL RESULTS

To gain insight in the algorithm behavior and to perform a preliminary assessment of the algorithm runtime behavior, we have implemented it in the SolBCN framework which can be downloaded under a GNU General Public License (see [19]).

The tests have been conducted on a standard desk computer with a Pentium IV at 3GHz processor and 1GB of core memory. The algorithm is implemented in Java using the Sun JDK. The tests were planned as follows. Two datasets were defined:

1. D_1: A set of 1000 randomly generated geometric constraint graphs with sizes ranging from 3 to 200 vertices. All the graphs were well-constrained but not necessarily solvable by the tree decomposition approach.
2. D_2: A set of 1000 randomly generated of geometric constraint graphs with sizes ranging from 3 to 200 vertices. All the graphs were under-constrained but not necessarily tree-decomposable.

We implemented four versions of the decomposition algorithm:

1. A_1: this algorithm recursively applies a set of predefined rules to discover hinges that split the graph, as reported in [5, 13, 16].
2. A_2: In this version first vertices of degree two are removed. Then algorithm A_1 is applied.
3. A_3: This version is an improvement of A_2 with specific treatment for 0-connected and 1-connected graphs.
4. A_4: is the algorithm presented in this work.

Let SG_s denote the set of graphs G such that $s = |V(G)|$. Notice that $3 \leq s \leq 200$. We applied each algorithm version to each dataset. For each algorithm and each graph in a

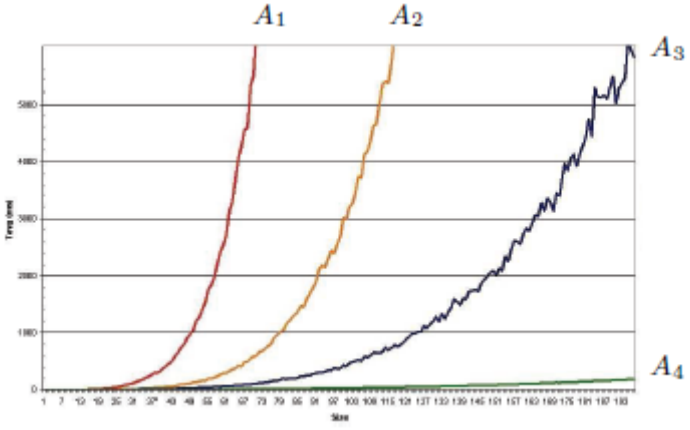

Figure 19: Behavior of the algorithms A_1, A_2, A_3, and A_4 on the dataset D_1.

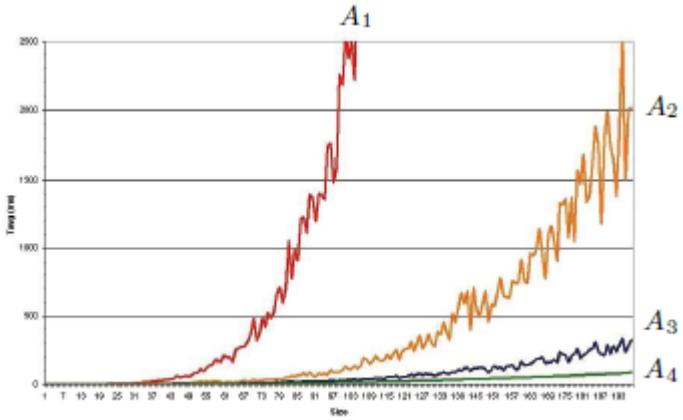

Figure 20: Behavior of the algorithms A_1**,** A_2**,** A_3**,and** A_4 **on the dataset** D_2**.**

data set, we recorded the algorithm runtime t(G). Then for each graph size s, we averaged the runtime values as

$$T(s) = \frac{1}{|SG_s|} \sum_{\forall G \in SG_s} t(G)$$

The results yielded by these tests are represented in Figure 19 for dataset D_1 and in Figure 20 for dataset D_2.

These results show that for both datasets the algorithm A_4 introduced in this paper exhibits a noticeable improved behavior. For graphs G with |V (G)| ≈ 200, the runtime for the algorithm A_4 is of about 200ms what allows interactive use in the SolBCN framework.

6. SUMMARY AND FUTUREWORK

We have presented a new algorithm to compute a decomposition tree of under- or wellconstrained treedecomposable geometric constraint graphs. The algorithm is based on directly computing sets of hinges by inspecting circuits of a particular planar embedding of the geometric constraint graph. The algorithm has been implemented and the empirical tests show a significant improvement with respect to treedecompositions based on applying a predefined set of rules.

In the near future we plan to work following three main directions. The first one is to prove the algorithm correctness. Then, since experimental results suggest a quadratic behavior, we plan to carry out an in depth study of the algorithm complexity. The current implementation considers only sets of hinges including three vertices. This means that the algorithm decomposes only quadratically solvable graphs. We plan to extend the method scope by considering split subgraphs where hinges share with the circuits more than three vertices.

7. ACKNOWLEDGMENTS

This research has been partially funded by the Spanish Ministerio de Educación y Ciencia and by FEDER under grant TIN2007-67474-C03-01.

8. REFERENCES

[1] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Computational Geometry. Springer, Berlin, 1997.
[2] C. B. Durand. Symbolic and numerical techniques for constraint solving. PhD thesis, Computer Science department, Purdue University, 1998.
[3] S. Even. Graph Algorithms. Computer Science Press, Potomac, Md., 1979.
[4] I. Fáry. On straight line representations of planar graphs. Acta Sci. Math., 11:229–233, 1948.
[5] I. Fudos and C. M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. ACM Transactions on Graphics, 16(2):179–216, 1997.
[6] X.-S. Gao, Q. Lin, and G.-F. Zhang. A c-tree decomposition algorithm for 2d and 3d geometric constraint solving,. Computer-Aided Design, 38(1):1–13, 2005.
[7] C. Hoffmann, A. Lomonosov, and M. Sitharam. Decomposition plans for geometric constraint systems, Part I: Performance measures for CAD. Journal of Symbolic Computation, 31(4):367–408, Apr. 2001.
[8] C. Hoffmann, A. Lomonosov, and M. Sitharam. Decomposition plans for geometric constraint systems, Part II: New algorithms. Journal of Symbolic Computation, 31(4):409–427, Apr. 2001.
[9] C. M. Hoffmann and R. Joan-Arinyo. A brief on constraint solving. Computer-Aided Design and Applications, 5(2):655–663, 2005.
[10] J. Hopcroft and R. Tarjan. Algorithm 447: efficient algorithms for graph manipulation. Commun. ACM, 16(6):372–378, 1973.
[11] C. Jermann, G. Trombettoni, B. Neveu, and P. Mathis. Decomposition of geometric constraints systems: A survey. International Journal of Computational Geometry and Applications, 23(7):1–35, March 2006.

[12] R. Joan-Arinyo, A. Soto-Riera, M. Tarrés-Puertas, and S. Vila-Marta. Decomposition of geometric constraint graphs based on computing fundamental circuits. Research report LSI-07-31-R, Department Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, 2007.
[13] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana. Revisiting decomposition analysis of geometric constraint graphs. Computer-Aided Design, 36:125–140, 2004.
[14] G. L. Miller and V. Ramachandran. A new graph triconnectivity algorithm and its parallelization. Combinatorica, 12(1):53–76, 1992.
[15] T. Nishizeki and N. Chiba. Planar Graphs. Theory and Algorithms. Dover Publications, Inc., Mineola, NY, USA, 2008.
[16] J. Owen. Algebraic solution for geometry from dimensional constraints. In SMA'91: Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications, pages 397–407, New York, NY, USA, 1991. ACM, ACM Press.
[17] S. Skiena. Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica. Addison-Wesley, Reading, MA, USA, 1990.
[18] A. Soto-Riera. Geometric Constraint Solving in 2D. PhD thesis, Departament de Llenguatges i Sistemes Informàtics, UPC, 1998.
[19] A. Soto-Riera, S. Vila-Marta, D. Silva, and M. Freixa. The SolBCN geometric constraint solver. Source code from http://floss.lsi.upc.edu, 2007. Under GNU-GPL license.
[20] K. Thulasiraman and N. Swamy. Graphs: Theory and Algorithms. John Wiley & Sons, 1992.
[21] H. Whitney. Non-separable and planar graphs. Proceedings of the National Academy of Sciences of the United States of America, 17(2):125–127, February 1931.