Decomposition of Geometric Constraint Graphs Based on Computing Fundamental Circuits

R. Joan-Arinyo, T. Soto, M. Tarrés-Puertas, S. Vila
Grup d'Informàtica a l'Enginyeria
Universitat Politècnica de Catalunya
Diagonal 647, 08028 Barcelona, Catalunya

Abstract

The graph-based geometric constraint solving technique works in two steps. First the geometric problem is translated into a graph whose vertices represent de set of geometric elements and whose edges are the constraints. Then the constraint problem is solved by decomposing the graph into a colection 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 biconnected graphs representing geometric constraint problems. The algorithm is based on recursively splitting the graphs through terns of vertices located on fundamental circuits of the graph. Preliminar experiments suggest that the algorithm runtime is at worst quadratic with the total number of vertices in the graph. In a case study we illustrate how the algorithm works.

Keywords Geometric constraint solving, Graph decomposition.

1 Introduction

Geometric models are data structures designed to represent physical properties of objects. The main properties encoded include geometric shape and topological properties of physical objects.

Computer Aided Design (CAD) systems are software applications built to help industrial designers during 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. Contemporary CAD/CAM systems capture the design intent partially by defining functional relationships between dimensional variables, that is, *parametric design* and *geometric constraint-based design*. One example of comercial CAD based in parametric design is Proengineer, [1].

One of the issues found in parametric, geometric constraint based design is the *geometric constraint solving problem*. This problem 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.

We consider 2D constraint problems defined by a set of geometric elements like points, lines, line segments, circles and circular arcs, along with a set of constraints like distance, angle, incidence and tangency between any two geometric elements. The algorithms that solve geometric constraint problems are named *solvers*. The reader is referred to [2, 7, 9, 15] for an extensive review of solving algorithms.

A geometric constraint problem can be coded as a *constraint graf* $G = (V, E)$, where the graph vertices V are the geometric elements with two degrees of freedom and the graph edges E are the geometric constraints, each canceling one degree of freedom.

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. The output is a *constructive plan*, that is, a sequence of basic steps that describe how to build a solution. Usually the basic steps are related to elemental ruler-and-compass operations.

Many attempts to provide general, powerful and efficient constructive graph-based techniques have been reported in the literature. For an extensive review refer to [4, 5, 6, 14].

Joan-Arinyo et al., [10, 11, 12, 19], defined the tree decomposition of a constraint graph. This concept has been specially useful from a theoretical point of view. Due to this contribution, the domains of Owen [14] and Fudos, [4], algorithms have been proved to be the same.

In this report we study the tree decomposition of a constraint graph from an algorithmic point of view. A recursive application of this algorithm yields a tree decomposition of the graph. Given a geometric constraint graph $G = (V, E)$, the problem we are facing is to efficiently compute the three hinges $\{v_1, v_2, v_3\} \subseteq V$ that induce a set decomposition of V.

This report is organized as follows. In Section 2 we review some basic concepts from graph theory. In Section 3 we present an introduction to geometric constraint graphs. Sections 4 and 5 introduce an algorithm to compute the hinges of a geometric constraint graph. Section 4 shows an algorithm to decompose 0-connected and 1-connected graphs. In Section 5 we present a decomposition algorithm for biconnected graphs. In Section 6 we illustrate how the algorithm works by following a case study. Finally, in Section 7 we give a summary and outline the future work.

2 Graph Concepts

In this section we recall some basic concepts and terminology of graph theory that will be used in the rest of the paper. Among the topics presented are graph decomposition in fundamental circuits and the depth first search algorithm. For an extensive treatment see the books of Even, [3], and Thulasiraman and Swamy, [17].

2.1 Basic Definitions

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 an edge e are called the *end vertices* of e and, when needed, will be denoted by $V(e)$. In general, $V(E)$ will denote the set of vertices of E. A graph can be represented by a diagram

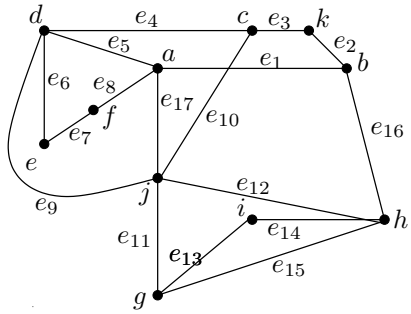

Figure 1: Graph example.

in which a vertex is symbolized by a dot and an edge by a line segment connecting two dots.

Example 1 *Figure 1 shows a graph* $G = (V, E)$ *where*

$$\begin{aligned} V &= \{a, b, c, d, e, f, g, h, i, j, k\} \\ E &= \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}\} \end{aligned}$$

$\Diamond$

The number of edges incident on a vertex v is called the *degree* of the vertex, and is denoted by $d(v)$. An *isolated vertex* is a vertex with degree 0.

Example 2 *In the graph of Figure 1,* $d(a) = 4$ *and* $d(e) = 2$. $\Diamond$

Consider the graph $G = (V, E)$. A subgraph of G is a graph $G' = (V', E')$ where

1. V' is a subset of V.
2. E' consists of edges (v_i, v_j) in E such that both v_i, v_j are in V'.

Consider the graph $G = (V, E)$ where for each vertex v_i in V, $d(v_i) \geq 1$. Let $V' \subset V$. Then the subgraph induced by V' in G is the graph $G' = (V', E')$ where E' consists on all edges $v_i, v_j \in V'$ such that both v_i and v_j are in V'. An induced subgraph G' of G will be denote by $G' \subset G$.

According to Thulasiraman, [17], a *walk* in a graph $G = (V, E)$ is a finite alternating sequence of vertices and edges $[v_0, e_1, v_1, e_2, \ldots, v_{k-1}, e_k, v_k]$ beginning and ending with vertices such that v_{i-1} and v_i are the end vertices of edge e_i, $1 \leq i \leq k$. A walk can be considered as a finite sequence of vertices $[v_0, v_1, v_2, \ldots, v_k]$ such that (v_{i-1}, v_i), $1 \leq i \leq k$, is an edge in the graph G. This walk is usually called a $v_0 - v_k$ walk where v_0 and v_k are the first and last vertices of the walk. All other vertices are internal to the walk. Note that in a walk, edges and vertices can appear more than once. A walk is *open* if its end vertices are distinct otherwise it is *closed*.

Example 3 *In the graph in Figure 1, the sequence* $[d, e_9, j, e_{17}, a, e_8, f, e_8, a, e_5, d]$ *is a closed walk, whereas the sequence* $[j, e_{17}, a, e_8, f, e_8, a, e_5, d]$ *is an open walk.* $\Diamond$

A walk is a *trail* if all its edges are distinct. A trail is open if its end vertices are distinct otherwise it is closed.

Example 4 *The walk* $[j, e_{17}, a, e_5, d, e_6, e, e_7, f, e_8, a, e_1, b]$ *in Figure 1 is an open trail, whereas the sequence* $[j, e_{17}, a, e_5, d, e_6, e, e_7, f, e_8, a, e_1, b, e_{16}, h, e_{12}, j]$ *is a closed trail.* $\Diamond$

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.

Example 5 *In Figure 1 the open trail* $[j, e_{17}, a, e_5, d, e_4, c]$ *is a* $j - c$ *path, whereas the closed trail* $[j, e_{17}, a, e_5, d, e_4, c, e_{10}, j]$ *is a circuit.* $\Diamond$

2.2 Operations on Graphs

In this section we introduce a few operations involving graphs.

Consider two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. The *union* of G_1 and G_2, denoted as $G_1 \cup G_2$, is the graph $G_3 = (V_1 \cup V_2, E_1 \cup E_2)$.

The *intersection* of G_1 and G_2, denoted as $G_1 \cap G_2$, is the graph $G_3 = (V_1 \cap V_2, E_1 \cap E_2)$.

The *ring sum* of two graphs G_1 and G_2, denoted as $G_1 \oplus G_2$, is the graph $G_3(E, V)$ such that edges in E belong to either E_1 or E_2 but not to both and whose vertices are $V(E)$.

Consider the graph $G = (V, E)$ and let $v_i \in V$. Then *vertex removal* $G - v_i$ is the induced subgraph of G on the vertex set $V - v_i$; that is, $G - v_i$ is the graph obtained after removing vertex v_i from V and all the edges in E incident on v_i.

Consider the graph $G = (V, E)$ and let $e_i \in E$. Then *edge removal* $G - e_i$ is the subgraph of G that results by removing the edge e_i from E. Note that the end vertices of e_i are not removed from G.

The removal of a set of vertices or edges from a graph G is defined as the removal of single vertices or edges in succession.

2.3 Connectivity

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.

Example 6 *Figure 2 shows a disconnected graph. The set of vertices in each connected component are* $V_1 = \{a, b, k, c, d, e, f\}$ *and* $V_2 = \{g, h, i\}$. $\Diamond$

Let $G = (V, E)$ be a connected graph. We say that a vertex $v \in V$ is an *articulation vertex* if $G - v$ is disconnected.

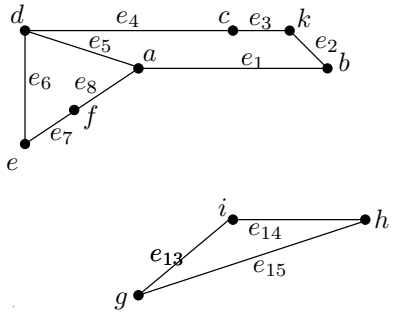

Figure 2: A disconnected graph.

Theorem 7 *Let $G = (V, E)$ be a connected graph. 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.*

Proof
See Thulasiraman and Swamy, [17]. □

A *non-separable* or *biconnected* graph $G = (V, E)$ has no articulation vertices, otherwise it is *separable*. 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 beconnected graph $G = (V, E)$, given a pair of vertices $u, v \in V$ with $u \neq v$, there are, at least, two disjoint paths $u - v$ paths.

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 or 0-connected graph, G, $\kappa(G) = 0$. For a connected or 1-connected graph, G, $\kappa(G) \geq 1$. A separable graph G has $\kappa(G) = 1$. A biconnected or 2-connected graph G has $\kappa(G) \geq 2$. In a similar way a graph G with $\kappa(G) \geq 3$ is called triconnected or 3-connected graph. Biconnected graphs can be decomposed into triconnected components.

2.4 Trees, Fundamental Circuits and Fundamental Cutsets

In this section we introduce trees, cutsets and several results associated with them. We also point out the relationship between trees, cutsets and circuits.

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 uin V. The *cospanning tree* T^* of a spanning tree T for a graph G is the subgraph of G whose vertices are those of G and whose edges are exactly those edges of G that are not in T. The edges of a spanning tree T are called the *branches* of T, and the edges in the cospanning tree T^* are called *chords*. Notice that a cospanning tree may not be connected.

Example 8 *Figure 3 shows a graph G, a spannig tree T and the corresponding cospanning tree T^*.* ◊

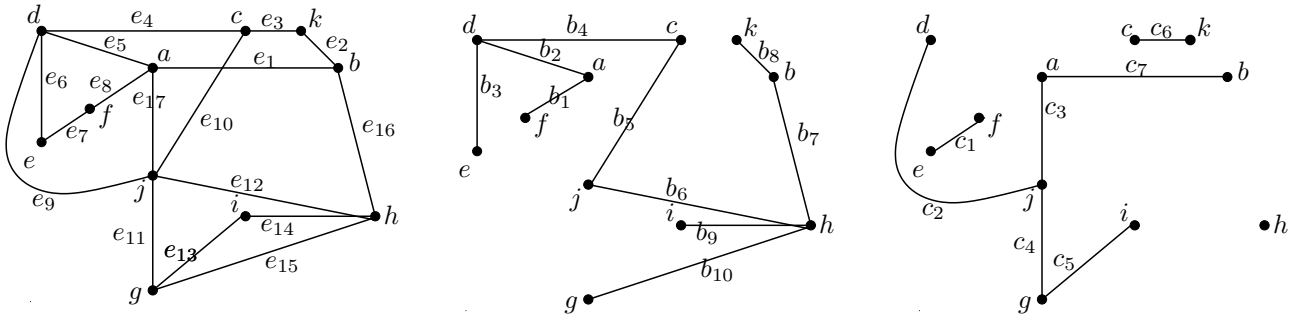

Figure 3: G, T and T^*.

Theorem 9 *Let $G = (V, E)$ a graph with $|V| = n$ and $G' \subset G$. G' is said to be a* spanning tree *of G if and only if G' is acyclic, connected, and has $n - 1$ edges.*

Proof
See Thulasiraman and Swamy, [17]. □

Let $G = (V, E)$ a graph with $|V| = n$ and $|E| = m$. Let T be a *spanning tree* to G. Let $b_1, \ldots, b_{n-1}$ be the *branches* and $c_1, \ldots, c_{m-n+1}$ the *chords* of T. The graph resulting from adding to T the chord 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 the *set of fundamental circuits*. An important feature of a fundamental circuit C_i is that it contains exactly one chord, namely, chord c_i. Furthermore, this chord is not present in any other fundamental circuit respect to T. Because of these properties, given a graph G and a spanning tree for it T, every circuit in G can be expressed as the *ring sum* of some fundamental circuits.

Example 10 *Consider the graph G and the spanning tree T in Figure 3. The fundamental circuits are the following*

$$\begin{aligned} C_1 &= [e, d, a, f] \\ C_2 &= [d, c, j] \\ C_3 &= [a, d, c, j] \\ C_4 &= [j, h, g] \\ C_5 &= [g, h, i] \\ C_6 &= [c, j, h, b, k] \\ C_7 &= [a, d, c, j, h, b] \end{aligned}$$

Every fundamental circuit contains exactly one chord that is not present in any other fundamental circuit. For example, chord (e, f) in C_1. The set of fundamental circuits are depicted in Figure 4. ◊

A *cutset* S of a connected graph G is a minimal set of edges of G such that its removal disconnects G, that is, the graph $G - S$ is disconnected.

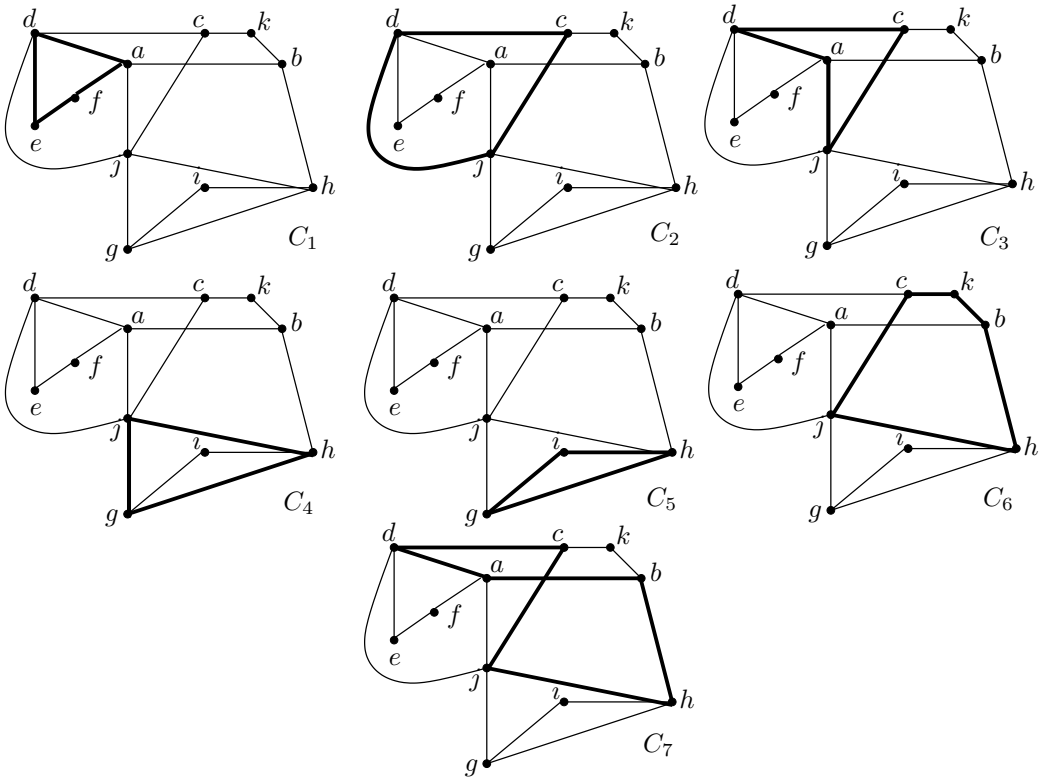

Figure 4: Fundamental circuits for the graph in Figure 3.

Example 11 *Consider the graph Figure 5 and the subset of edges* $S = \{e_1, e_3, e_7, e_{10}\}$. *The removal of* S *from* G *results in a disconnected graph. Furthermore, the removal of any proper subset of* S *cannot disconnect* G. *Thus* S *is a cutset of* G. $\Diamond$

We now define the concept of a cut, which is closely related to that of a cutset. Consider a connected graph $G = (V, E)$. Let V_1 and V_2 be two subsets such that $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$. Then the set S of all those edges of G having one end vertex in V_1 and the other in V_2 is called a *cut* of G. This is denoted by $\langle V_1, V_2 \rangle$.

Example 12 *Let* G *be the graph shown in the Figure 1. If* $V_1 = \{a, b, k, c, d, e, f, j\}$ *and* $V_2 = \{g, h, i\}$, *then the cut* $\langle V_1, V_2 \rangle$ *is the set of edges* $\{e_{11}, e_{12}, e_{16}\}$. $\Diamond$

A cut $\langle V_1, V_2 \rangle$ of G is the minimal set of edges of G whose removal disconnects G into two

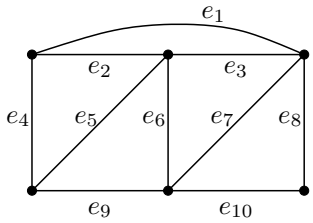

Figure 5: A graph and the cutset $\{e_1, e_3, e_7, e_{10}\}$.

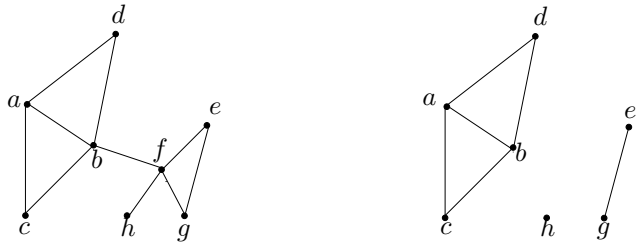

Figure 6: Left) Graph $G(V, E)$. Right) Subgraph induced by $V - f$.

subgraphs G_1 and G_2 being G_1 and G_2 the induced subgraphs of G by the vertex sets V_1 and V_2 respectively. If both G_1 and G_2 are connected, then $\langle V_1, V_2 \rangle$ is also the minimal set of edges disconnecting G into exactly two components. Then, by definition,$\langle V_1, V_2 \rangle$ is a cutset of G.

Theorem 13 *A cut in a connected graph G is either a cutset or a union of edge-disjoint cutsets of G.*

Proof
See Thulasiraman and Swamy, [17]. □

Theorem 14 *The set of edges incident on a vertex v in a connected graph G is a cutset of G if and only if v is not an* articulation vertex *of G.*

Proof
See Thulasiraman and Swamy, [17]. □

Example 15 *Consider the separable graph $G = (V, E)$ with the articulation vertex f. shown in Figure 6. The subgraph of G induced by the vertex set $V - f = \{a, b, c, d, e, h, g\}$ consists of three components. Thus the edges incident to the articulation vertex f are not a cutset of G.* ◊

A spanning tree of a connected graph can be used to obtain a set of fundamental cutsets of the graph. Let T be a spanning tree to the connected graph G. Let b be a branch of T. Now, the removal of branch b disconnects T into exactly two components T_1 and T_2. Let V_1 and V_2, respectively, denote the vertex sets of T_1 and T_2. V_1 and V_2 together contain all the vertices of G. Let G_1 and G_2 be the induced subgraphs of G on the vertex sets V_1 and V_2. It can be seen that T_1 and T_2 are, respectively, spanning trees of G_1 and G_2. Hence, $\langle V_1, V_2 \rangle$ is a cutset of G. This cutset is known as the *fundamental cutset* of G with respect to the branch b of the spanning tree T to G. The set of the $n - 1$ distinct cutsets defined by the branches of a spanning tree T of a connected graph G is known as the set of *fundamental cutsets* of G with respect to the spanning tree T. The cutset $\langle V_1, V_2 \rangle$ contains exactly one branch, namely, the branch b of T. All the other edges are chords of T. This follows from the fact that $\langle V_1, V_2 \rangle$ does not contain any edge of T_1 or T_2. Furthermore, branch b is not present in any other fundamental cutset with respect to T. Because of these properties, every fundamental cutset of a graph G can be expressed as the ring sum of the fundamental cutsets of G with respect to the spanning tree T to G. The edge set of no fundamental cutset can be expressed as the ring sum of the edge sets of some or all of the remaining fundamental cutsets.

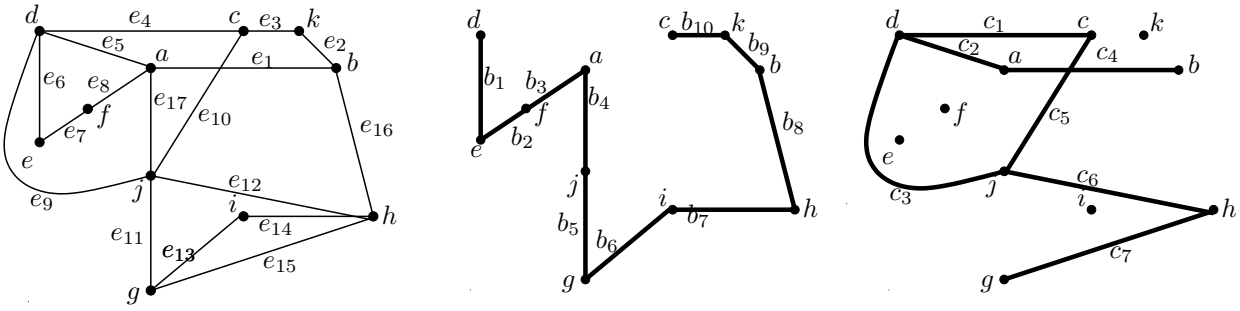

Figure 7: Left) Graph. Middle) A DFS spanning tree. Right) The Cospanning tree.

Example 16 *For example, the graph shown in Figure 3, have the following fundamental cutsets:*

$$\begin{aligned}
FC_1 &= \{(a,f),(f,e)\} \\
FC_2 &= \{(a,d),(f,e),(a,j),(a,b)\} \\
FC_3 &= \{(d,e),(e,f)\} \\
FC_4 &= \{(d,c),(d,j),(a,j),(a,b),(d,c)\} \\
FC_5 &= \{(c,j),(c,k),(a,j),(d,j),(a,b)\} \\
FC_6 &= \{(j,h),(c,k),(a,b),(c,k),(j,g)\} \\
FC_7 &= \{(h,b),(b,a),(k,c)\} \\
FC_8 &= \{(b,k),(k,c)\} \\
FC_9 &= \{(h,i),(i,g)\} \\
FC_{10} &= \{(h,g),(i,g),(j,g)\}
\end{aligned}$$

$\Diamond$

2.5 Depth-First Search

The Depth-First Search (DFS) algorithm is a technique that allows to efficiently visit vertices and edges in a graph and serves as an skeleton around which many other efficient graph algorithms can be built. See [8, 16].

Assume we are given a finite connected graph $G = (V, E)$. Starting in one of the vertices we want to walk along the edges, from vertex to vertex and visit all the vertices. The start vertex v is called *the root of the DFS*. DFS terminates when the search returns to the root and all the vertices have been visited. When selecting the next vertex to explore, depth-first search selects a vertex that has never been explored and is connected by an edge to the most recently explored vertex. DFS partitions the edges of G into edges visited (branches) and edges not visited (chords). The branches form a spanning tree T to G. The chords form a cospanning tree T^* to G.

Example 17 *Figure 7 shows a graph $G = (V, E)$, a spanning tree computed by the DFS with vertex d as the root and, the corresponding cospanning tree.* $\Diamond$

2.6 Graph Embedding

Since planar graphs play an important role in this work, in this section we introduce the main concepts related to graph planarity.

Consider a graph $G = (V, E)$. We say that G is *embeddable in a surface* S if G can be drawn in S in such a way that

1. Each vertex is represented by a point $p \in S$.
2. Each edge is represented by a continous curve $c \in S$ connecting the two points which represent its end vertices and,
3. No two curves share any point except the vertices.

Such a drawing is called an *embedding of* G *in* S. A graph G embedded in the Euclidean plane is said to be *planar*.

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

2.7 Components and Bridges

Following Even, [3], this section explains how a non-separable graph can be decomposed into one circuit and a set of bridges. This is achieved by first showing how an arbitrary set of vertices induces a graph decomposition. The result of this decomposition is a set of graph components. If a number of conditions hold in the decomposition, then the resulting components are bridges.

A graf G is called *non-separable* if it is connected, and if there are two graphs G_1 and G_2 each containing at least one edge, which form G if a vertex of one is made to coalesce with a vertex of the other. If G is not non-separable, it is separable, [20].

Let $G = (V, E)$ be a non-separable graph and let $S \subseteq V$. Consider the partition of the set $V - S$ into classes such that two vertices are in the same class if and only if there is a path connecting them which does include any vertex of S. Let $\equiv$ denote the relationship defined and assume that $V - S/_{\equiv} = \{K_1, \ldots, K_m\}$. Each class K_i defines a *non-singular component*, $H = (V', E')$ with $H \subseteq G$, as follows:

- $V' \supset K_i$.
- V' includes all the vertices of S which are connected by an edge to a vertex of K_i in G.
- $E' \subseteq E$ contains all edges of G which have at least one end vertex in K_i.

An edge (u, v) where both $u, v \in S$ defines a *singular component* $H = (\{u, v\}, \{(u, v)\})$. Therefore a set of vertices S induces a graph decomposition into a set of singular and non-singular components $\{H_1, \ldots, H_n\}$.

Two components share no edges, and the only vertices they can share are vertices of S. The vertices of a component $v \in V(H)$ such that $v \in S$ are called *attachments*.

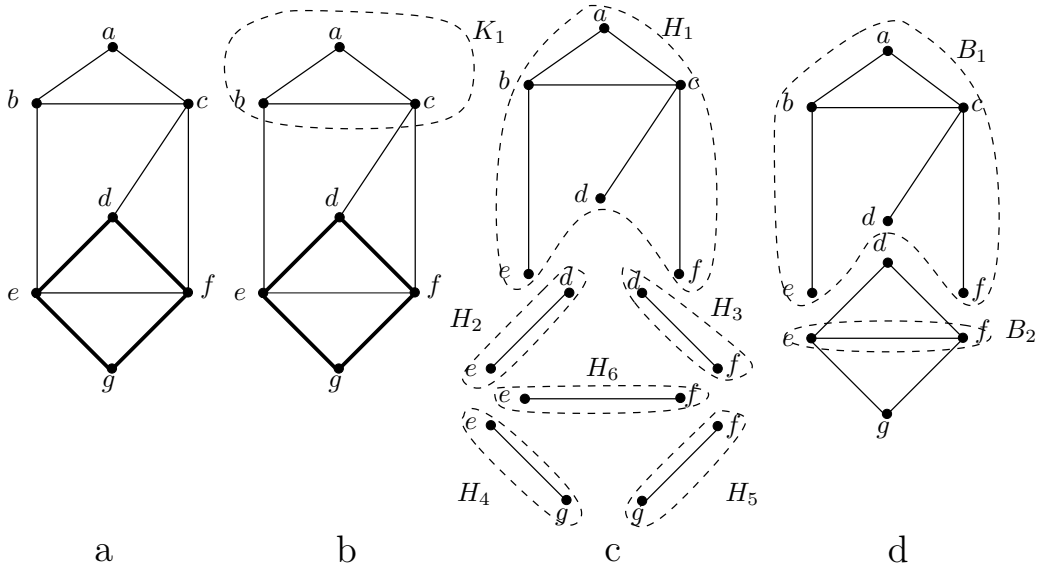

Figure 8: Graph, components and bridges.

Definition 18 *Let S be the set of vertices of a circuit in G. Let H be the set of components of G induced by S. A* bridge*, B_i, is a component $H_i \in H$ whose edges do not belong to S.*

A bridge is said to be singular if the corresponding component is singular otherwise it is non-singular.

Example 19 *Consider the graph $G = (V, E)$ given in Figure 8a and let $S = [d, e, g, f] \in V$ shown in bold. Since $V - S = \{a, b, c\}$, there is just one induced class $K_1 = \{a, b, c\}$ shown in Figure 8b. The components, shown in Figure 8c, are*

$$\begin{aligned}
H_1 &= (\{a,b,c,e,d,f\}, \{(a,b),(a,c),(b,c),(b,e),(c,d),(c,f)\}) \\
H_2 &= (\{d,e\}, \{(d,e)\}) \\
H_3 &= (\{d,f\}, \{(d,f)\}) \\
H_4 &= (\{e,g\}, \{(e,g)\}) \\
H_5 &= (\{g,f\}, \{(g,f)\}) \\
H_6 &= (\{e,f\}, \{(e,f)\})
\end{aligned}$$

H_1 is a non-sigular component, and H_i, $2 \leq i \leq 6$, are singular components.

Components H_1 and H_6 are bridges, shown in Figure 8d respectively as B_1 and B_2. Attachments of B_1 are $\{e, d, f\}$, and attachments of B_2 are $\{e, f\}$. B_2 is a singular bridge. $\Diamond$

Theorem 20 *Let B be a bridge with attachments a_1, a_2 and a_3. There exists a vertex $v \in B$, that is not an attachment for which there are three vertex disjoint paths in B: $v - a_1$, $v - a_2$ and $v - a_3$.*

Proof
See Even, [3]. □

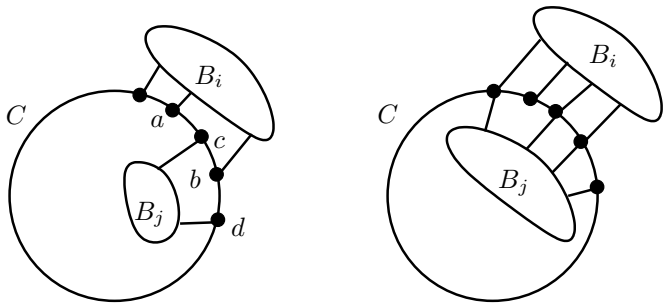

Figure 9: Interlacing attachments.

Example 21 *Consider the graph in Example 19. The attachments of bridge B_1 are $\{e, d, f\}$. Vertex $a \in B_1$ is not an attachment and you can find the paths $\{a, b, e\}$, $\{a, c, d\}$ and $\{a, c, f\}$.* $\Diamond$

2.8 Planar Graphs and Bridges

This section is devoted to the relationships between planarity and bridges. First, we introduce the concept of bridge interlacement. Then we consider the planar embedding of bridges.

Let $G = (V, E)$ be a non-separable graph and C a circuit in G. Let $B_1, \ldots, B_k$ be the bridges of G with respect to C. We say that B_i and B_j *interlace* if at least one of the following conditions holds:

1. There are two attachments a and b of B_i and two attachments c and d of B_j such that all four attachments are distinct and they appear in C in the sequence $[a, c, b, d]$.
2. There are three attachments common to B_i and B_j.

Figure 9 illustrates these conditions.

Example 22 *Consider the graph in Figure 10 and the circuit $C = [b, k, c, d, e, f, a]$ shown in bold line. The bridges of the graph are $B_1 = \{d, c, b\}$, $B_2 = \{d, a, c, b\}$ and $B_3 = \{d, a\}$. B_1 and B_2 interlace because they have in common three attachments.* $\Diamond$

For each bridge B_i, consider the subgraph $C \cup B_i \subseteq G$. If any of these subgraphs is not planar, then clearly G is not planar. Now, assume that all these subgraphs are planar. In every planar embedding of G, C divides the plane into two disjoint parts, one is bounded (inside) and the other is unbounded (outside). Two bridges that do not interlace can be embedded in any side of the circuit C. However, if two bridges interlace they cannot be embedded on the same side of C. Thus, in every planar embedding of G, C partitions the interlacing bridges into two sets: those which are drawn inside C and those which are drawn outside. No two bridges in the same set interlace. Notice that choosing the side where bridges are embedded does not matter.

Theorem 23 *If B_i, $1 \leq i \leq m$, is the set of bridges of a non-separable graph G with respect to a simple circuit C and the following two conditions are satisfied:*

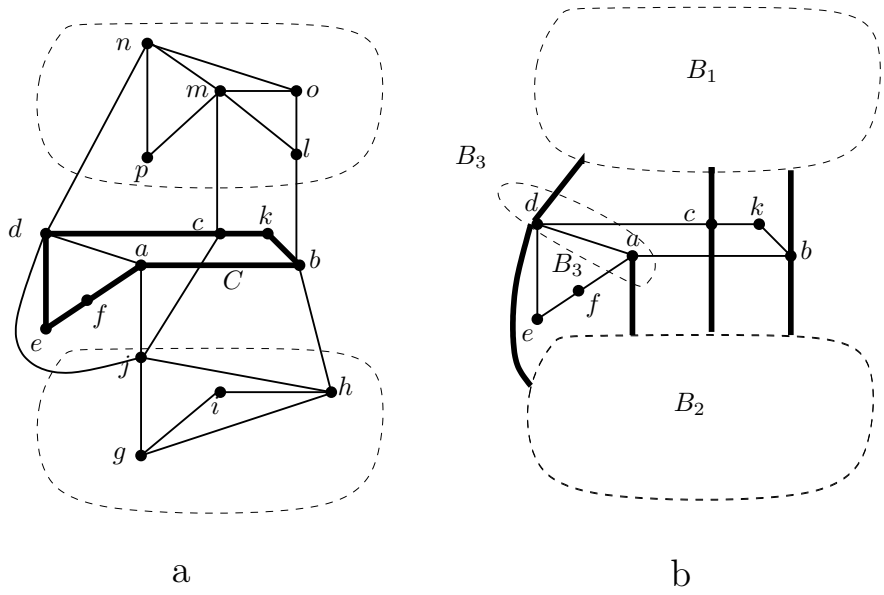

Figure 10: a) A graph with a cicuit C in boldline. b) Set of bridges induced B_1, B_2 and B_3.

1. *For every B_i, $C \cup B_i$ is planar.*
2. *No two bridges interlace,*

then $C \cup (\bigcup_{1 \leq i \leq m} B_i)$, has a planar embedding in which all the bridges are inside (or outside) of C.

Proof
See Even, [3]. □

2.9 Collapsed Graphs

We start recalling an standard concept from graphs field illustrated in Figure 11.

Definition 24 A star graph *is a connected graph $G = (V, E)$ with one vertex $v \in V$, called* center*, whose degree is $d(v) > 1$ and such that for each $v_i \in V$ with $v_i \neq v$, $d(v_i) = 1$.*

Star graphs are planar and are characterized by just listing the center and the set of vertices.

Now consider a graph $G = (V, E)$ whose bridges are $B = \{B_1, \ldots, B_m\}$. Let $B_i = (V_i, E_i)$ be a bridge in B with attachments $\{a_1, \ldots, a_n\} \subseteq V_i$. We define the star graph $S_i = (V_{S_i}, E_{S_i})$ such that $V_{S_i} = \{x, a_1, \ldots, a_n\}$ where x is a new vertex and, $E_{S_i} = \{(x, a_j), 1 \leq j \leq n\}$.

Definition 25 *Let G be a graph in the conditions described above. The* collapsed graph *of G is the graph resulting from replacing each bridge B_i with the corresponding star graph S_i.*

Example 26 *Figure 12 shows the collapsed graph corresponding to the graph in Figure 10. Star graphs are $S_1 = \{x_1, d, c, b\}$, $S_2 = \{x_2, d, a, b\}$ and $S_3 = \{x_3, a, d\}$.* ◊

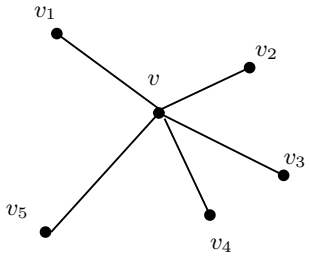

Figure 11: Star graph.

Notice that star graphs S_i and S_j in the collapsed graph interlace if and only if bridges B_i, B_j interlace in graph G.

2.10 Merged Graphs

In this section we introduce the concept of *merged graph*.

Let G' be the graph resulting from collapsing the bridges in graph $G = (V, E)$. Let S_i and S_j be two stars in G'. We define the merging of S_i and S_j as a new star graph S_{ij} such that the center is a new vertex and whose attachments are the union of the attachments in S_i and in S_j.

Example 27 *Figure 13 shows a collapsed graph and the corresponding merged graph. Notice that now merged stars S_{12} and S_{34} do not interlace.* $\Diamond$

The *merged graph* is computed as follows. Choose two interlacing stars, say S_i, S_j, and replace them by a new star S_{ij} resulting from merging S_i and S_j. Repeat this process until no two stars interlace.

Let G' be a merged graph computed from graph G with respect to the fundamental circuit C. G' is the union of the circuit C with the collapsed stars derived from G. Clearly, G' is a planar graph and Theorem 23 applies. Moreover, if $\{a, b, c\}$ are hinges of G which belong to

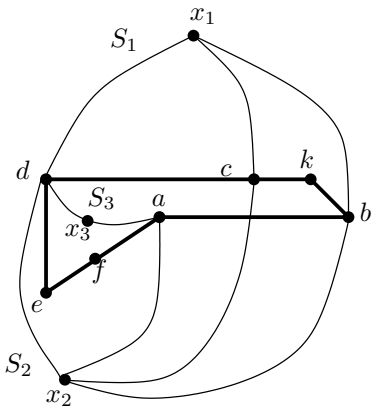

Figure 12: Collapsed graph corresponding to the graph in Figure 10b.

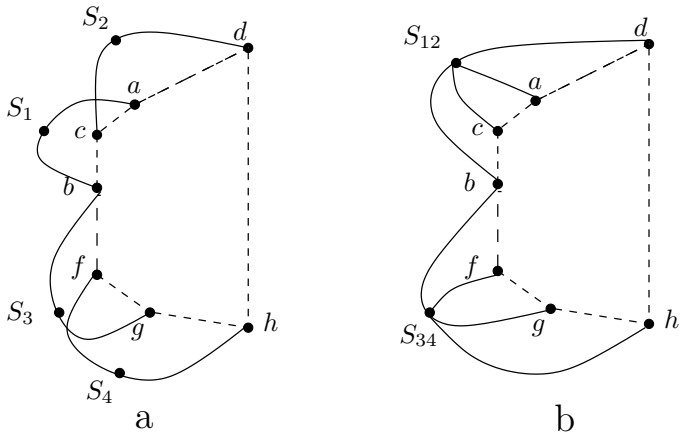

Figure 13: a) Collapsed graph. b) Merged graph.

the circuit C, vertices $\{a, b, c\}$ are also hinges for G' and belong to C.

3 Geometric Constraint Solving

In this chapter we define what is a geometric constraint problem and the associated geometric constraint graph. We also introduce the decomposition analysis of the constraint graph. Decomposition is a technique commonly used to analyze geometric constraint graphs in geometric constraint solving. For more information we refer the reader to the works by Hoffmann et al., [7], and R.Joan-Arinyo et al., [11].

3.1 The General Geometric Constraint Problem

A *general geometric constraints problem* consists of a set of geometric elements, a set of geometric constraints defined between them and a set of parameters, i.e. the values of the geometric constraints. In what follows, we consider that geometric elements are two dimensional points, lines, circles and arcs of circle with given radii. Geometric constraint is either a distance between two points, a distance between a point and a line, angle between two lines, tangencies perpendicularities and so on.

A geometric constraint problem can be characterized by means of a tuple (E, O, X, C) where

- E is the geometric space where the problem is embedded. E is usually Euclidean.
- O is the set of specific geometric objects which define the problem. They are chosen from a fixed repertoire including points, lines, circles and the like.
- X is a, possibly empty, set of variables whose values must be determined. In general, variables represent quantities with geometric meaning: distances, angles and so on. When the quantities are without a geometric meaning, for example, when they quantify technological aspects and functional capabilites, those variables are called *external*.

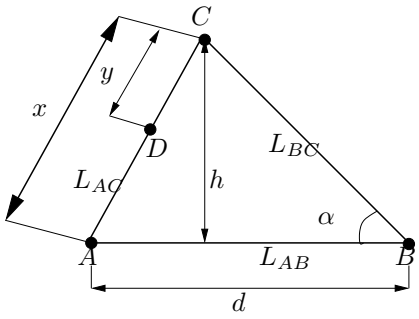

Figure 14: The general geometric constraint solving problem in 2D.

- C is the set of constraints. Constraints can be geometric or equational. Geometric constraints are relationships between geometric elements chosen from a predefined set, e.g., distance, angle, tangency, etc. The relationship (the distance, the angle,...) is represented by a tag. If the tag represents a fixed value, known in advance, then the constraint is called *valuated*. If the tag represents a value to be computed as part of solving the constraint problem, then the constraint is called *symbolic*. Equational constraints are equations some of whose variables are tags of symbolic constraints. The set of equational constraints can be empty.

The geometric constraint solving problem can now be stated as follows:

Given a geometric constraint problem (E, O, X, C),

1. Are the geometric elements in O placed with respect to each other in such a way that the constraints in C are satisfied? If the answer is positive, then
2. Given an assignment of values to the valuated constraints and external variables, is there an actual construction that satisfies the constraints and equations?

Example 28 *Figure 14 depicts a general geometric constraint solving problem in 2D. Problem components are defined as follows:*

- *The set of geometric elements,* $O = \{A, B, C, D, L_{AB}, L_{AC}, L_{BC}\}$.
- *The set of tags in the constraints,* $P = \{d, h, \alpha\}$.
- *The set of geometric variables,* $V_1 = \{x, y\}$.
- *The set of external variables,* $V_2 = \{v\}$.
- *The set of geometric constraints with fixed tags,* $C_1 = \{dpp(A, B) = d, dpl(C, L_{AB}) = h, ang(L_{AB}, L_{BC}) = \alpha\}$.
- *The set of geometric constraints with variable tags,* $C_2 = \{dpp(A, C) = x, dpp(C, D) = y\}$.
- *The set of equational constraints,* $C_3 = \{y = x \cdot v, v = 0.5cos(\alpha)\}$.

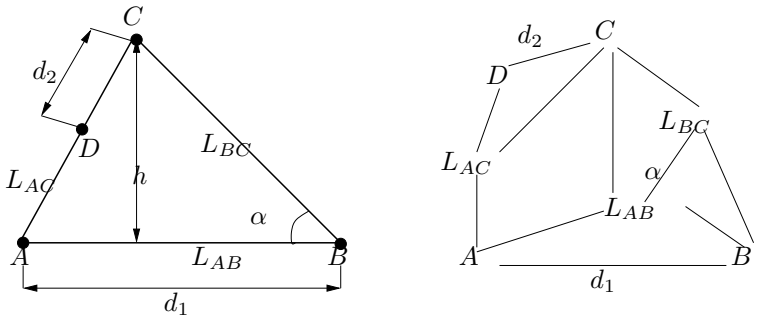

Figure 15: A geometric constraint problem and the associated constraint graph.

with $X = V_1 \cup V_2$ *and* $C = C_1 \cup C_2 \cup C_3$.

The meaning of the basic construction names is the usual: point defined by its coordinates, straight line given by and ordered pair of points. For example, L_{AB} *defines a line between two points* A *and* B*. Predicate names are self explanatory. The predicate* $dpp(A, B) = d$ *specifies a point-point distance,* $dpl(C, L_{AB}) = h$ *defines the signed perpendicular distance from a point to a straight line and,* $ang(L_{AB}, L_{BC}) = \alpha$ *denotes the angle between two straight lines.* $\Diamond$

3.2 The Basic Problem

The basic constraint problem only considers geometric elements and constraints whose tags are assigned a value. It excludes external variables, constraints whose tags must be computed, and equational constraints. So the basic problem is stated in the following way.

> Given a set O with n geometric elements and a set C with m geometric constraints defined on them
>
> 1. Is there a placement of the n geometric elements such that the m constraints are fullfilled? If the answer is positive,
> 2. Given an assignment of values to the m constraints tags, is there an actual construction of the n geometric elements satisfying the constraints?

In what follow we focus on the basic problem.

3.3 Geometric Constraint Solving and Graphs

A 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.

Example 29 *Figure 15 shows a geometric constraint problem and the associated constraint graph.* $\Diamond$

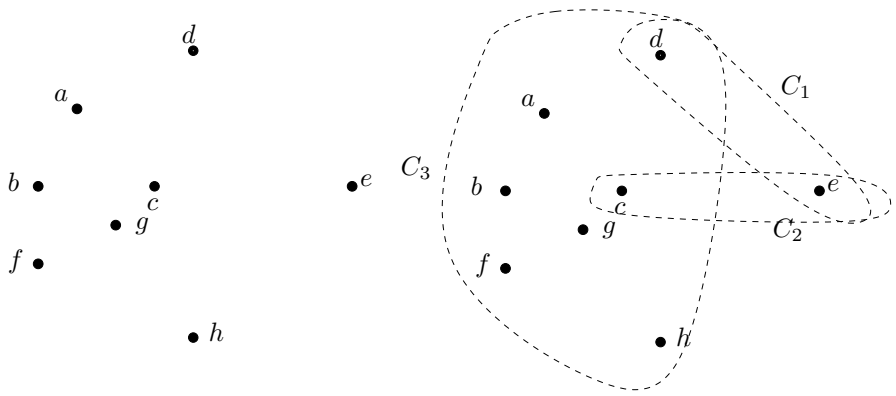

Figure 16: Set and set decomposition.

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, in this work graphs with exactly three vertices, are found to which standard equational solvers are applied. Therefore, devising feasible algorithms that efficiently decompose constraint graphs is paramount.

3.4 Decomposition of a Constraint Graph

In this section we first define the concepts of *set decomposition* and *tree decomposition*. Then we outline the algorithm used to compute tree decompositions.

3.4.1 Definitions

The concept of *set decomposition* refers to a way of partitioning a given abstract set. Next we define the concept of *tree decomposition* of a graph.

Definition 30 *Let C be a set with at least three different members, say a, b, c. Let $C_1, C_2, C_3 \subset C$. 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. $C_2 \cap C_3 = \{b\}$ *and*
4. $C_1 \cap C_3 = \{c\}$.

We say that vertices a, b, c are the *hinges* of the set decomposition, and C_1, C_2 and C_3 are *clusters*. Notice that a set decomposition is not necessarily unique.

Example 31 *Figure 16 shows a set and a set decomposition.* $\Diamond$

Next we define the concept of *set decomposition of a graph*.

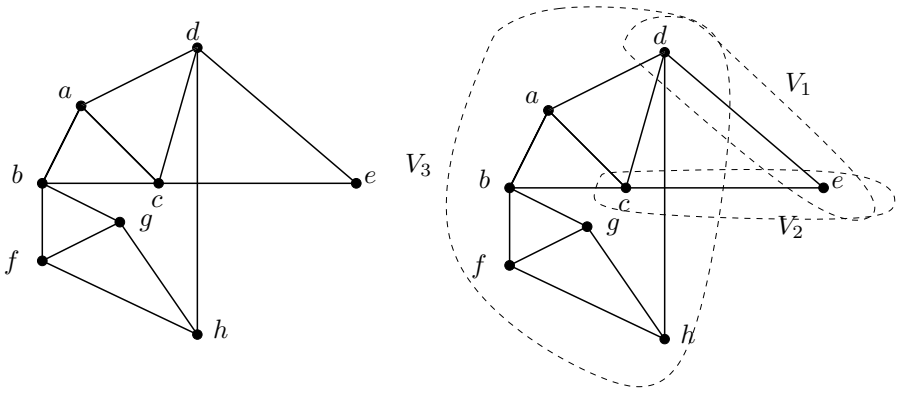

Figure 17: Set decomposition of a graph.

Definition 32 *Let $G = (V, E)$ be a graph and $V_1, V_2, V_3 \subseteq V$. Then V_1, V_2 and 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 $i, 1 \leq i \leq 3$.*

Example 33 *The concept of set decomposition for a graph G is illustrated in Figure 17.* $\Diamond$

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.

Example 34 *Figure 18 shows a graph $G = (V, E)$ and a set decomposition of V which is not a set decomposition of G because vertices incident to edge (e, b) do not belong to any set in the partition.* $\Diamond$

Next, we define the concept of *tree decomposition* of a graph.

Definition 35 *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,*

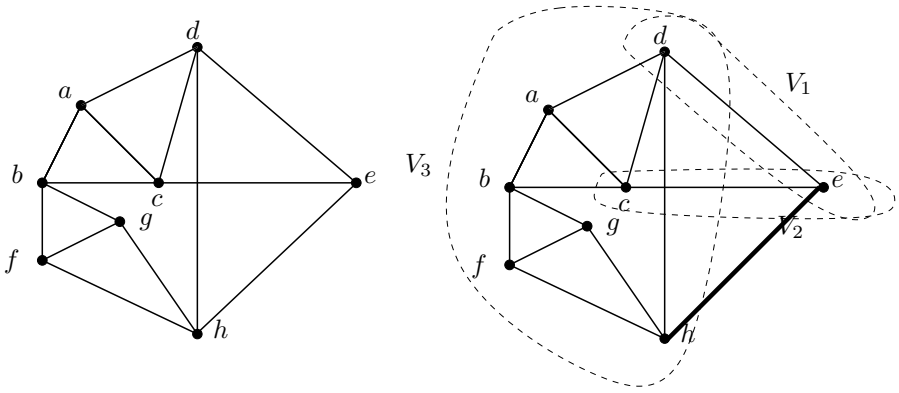

Figure 18: Set decomposition of a graph with a broken edge.

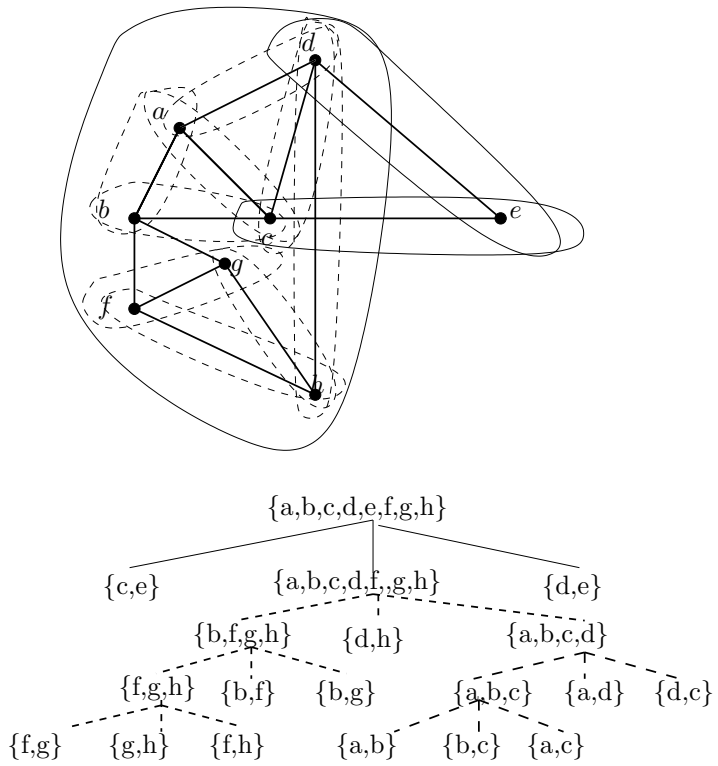

Figure 19: Collection of set decompositions of a graph.

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 will be called *tree decomposable graph*.

Example 36 *Figure 19 shows a collection of set decompositions recursively generated for the tree decomposable graph in Figure 17 and, the corresponding tree decomposition.* $\Diamond$

If a graph is tree decomposable, then any subgraph obtained by removing some of its edges is also tree decomposable. This is a useful property of tree decompositions we will make use of later on.

Theorem 37 *Let $G = (V, E)$ be a tree decomposable graph. For all $E' \subseteq E$, the subgraph of G with the set of edges E', $G' = (V, E')$, is tree decomposable.*

Proof
See Joan-Arinyo *et al.*[12]. $\square$

INPUT: constraint graph $G = (V, E)$ with $|V| \geq 3$
OUTPUT: a set of hinges $\{a, b, c\} \subseteq V$, if one exists

forevery $\{a, b, c\} \subseteq V$ **do**
    $V' = V - \{a, b, c\}$
    forevery $\{G'_1, G'_2, G'_3\}$ partition of V' **do**
        $G_1 = G'_1 \cup \{a, c\}$
        $G_2 = G'_2 \cup \{a, b\}$
        $G_3 = G'_3 \cup \{b, c\}$
        if $\forall e \in E : V(e) \subseteq G_1$ or $V(e) \subseteq G_2$ or $V(e) \subseteq G_3$ **then**
        return $\{a, b, c\}$
    endfor
endfor
return $\emptyset$

Figure 20: Finding hinges by means of exhaustive search.

3.5 Computing a Tree Decomposition

The goal now is to give an algorithm to compute a set decomposition of a constraint graph $G = (V, E)$. A recursive application of this algorithm yields a tree decomposition of a graph. In this section we just give a preliminary algorithm based on an exhaustive search of all the vertices of G in order to find hinges. In the next section we offer a more efficient approach.

A naive approach to computing hinges is to perform an exhaustive search. This amounts to considering different combinations of three different vertices of V until finding one tern that allows to split the graph G in three disjoint clusters G_1, G_2 and G_3. The algorithm is shown in Figure 20. Notice that the number of different sets with three vertices is $\binom{|V|}{3}$.

Notice that the algorithm fails if it does not find any triplet of vertices such that splits the graph. Clearly this means that the graph is no decomposable.

4 Computing Hinges in Trivial Cases

In this section we consider geometric constraint graphs with more than three vertices and 0-connectivity or 1-connectivity.

First, we present an algorithm to compute the hinges of a 0-connected constraint graph. Then we present an algorithm to compute the hinges of a 1-connected constraint graph. Finally, we give an algorithm for dealing with graphs with vertices of degree two. Although this is just a particular case, constraint graphs with vertices of degree two are in practice very usual.

4.1 Decomposition of 0-connected Graphs

Let $G = (V, E)$ be a 0-connected constraint graph with $|V| > 3$. Let $K = \{K_1, \ldots, K_n\}$ be the set of connected components of G. Since G is 0-connected, $|K| \geq 2$. To compute the

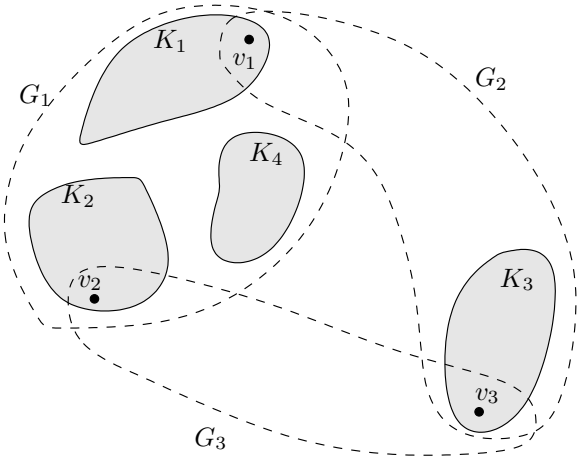

Figure 21: Decomposition of 0-connected graphs.

hinges of G we proceed as follows (refer to Figure 21).

1. Let G_1 be a non trivial subset of K containing at least two vertices $v_1, v_2 \in V$ with $v_1 \neq v_2$. Notice that G_1 must exists because $|V(G)| > 2$.
2. Let v_3 be an arbitrary vertex of $V - V(G_1)$.
3. Then v_1, v_2 and v_3 are hinges of G.

Figure 22 shows an efficent algorithm to decompose a 0-connected graph.

4.2 Decomposition of 1-connected Graphs

Let $G = (V, E)$ be a 1-connected constraint graph with $|V| > 3$. To compute the hinges of G we proceed as follows (see Figure 23):

1. Let v_1 be an articulation vertex of G. v_1 must exist because G is 1-connected.
2. Let $K = K_1, \ldots, K_m$, be a set of connected components of the graph $G_1 = G - v_1$. $|K| \geq 2$ because v_1 is an articulation vertex. Arbitrarily select two connected components of K, say $K_1 = (V_1, E_1)$ and $K_2 = (V_2, E_2)$. Let $v_2 \in V_1$ and $v_3 \in V_2$.
3. Then v_1, v_2 and v_3 are hinges of G.

Example 38 *Figure 24 shows a 1-connected graph with hinges* $\{c, e, g\}$. $\Diamond$

Figure 25 shows an efficent algorithm to decompose a 1-connected graph.

INPUT: 0-connected constraint graph $G = (V, E)$ with $|V| \geq 3$
OUTPUT:$\{v_1, v_2, v_3\}$, hinges of a set decomposition of G

Choose K, a connected component of G
if $|V(K)| > 1$ **then**
    Choose $v_1, v_2 \in V(K)$
    Choose $v_3 \in V - V(K)$
else
    /* K is a component with one vertex */
    Choose $v_1, v_2 \in V - V(K)$
    Choose $v_3 \in V(K)$
endif
return $\{v_1, v_2, v_3\}$

Figure 22: Decomposition of 0-connected graphs.

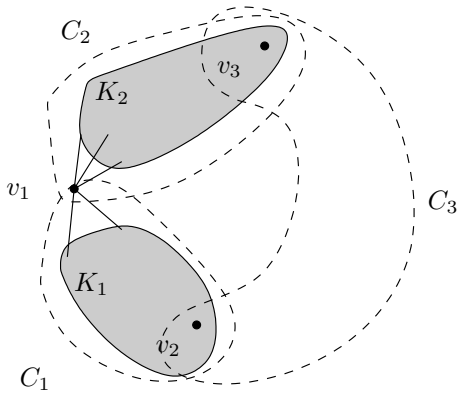

Figure 23: Decomposition of 1-connected graphs.

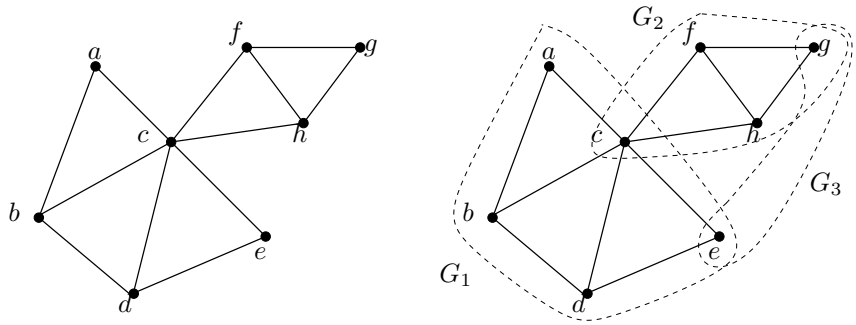

Figure 24: A 1-connected graph and hinges $\{c, e, g\}$.

INPUT: 1-connected constraint graph $G = (V, E)$ with $|V| \geq 3$
OUTPUT:$\{v_1, v_2, v_3\}$, hinges of G

Select one articulation vertex $v_1 \in V$
$G_1 = G - v_1$
Choose $K_1, K_2 \subset G_1$ two arbitrary connected components
Choose $v_2 \in V(K_1)$ and $v_3 \in V(K_2)$
return $\{v_1, v_2, v_3\}$

Figure 25: Decomposition of 1-connected graphs.

4.3 Decomposing Graphs with Degree 2 Vertices

In this section we show a way for decomposing a graph $G = (V, E)$ with connectivity $k \geq 1$ and with vertices of degree 2. Todd [18], reported on a method where graphs are subdivided by isolating vertices of degree two from their neighbors. This subdivision method is rather limited but can be satisfactorily combined with other techniques for decomposing graphs. Graphs with degree 2 vertices can be decomposed as follows.

1. Let $G = (V, E)$ be a graph, with $v \in V$ a vertex with degree 2.
2. Choose two different vertices v_1 and v_2 from V such that $(v, v_1) \in E$ and $(v, v_2) \in E$.
3. Then G, $(v, v_1, v_2$ are hinges with clusters $G_1 = \{v, v_1\}$, $G_2 = \{v, v_2\}$, $G_3 = V - \{v\}$. Decomposition is shown in Figure 26.

This decomposition technique can be efficiently implemented if the graph is stored as an adjacency list where the set of vertices with degree 2 can be easily identified.

5 Computing Hinges in 2-connected Graphs

In this section we first study the relationship between hinges and fundamental circuits in biconnected graphs. Then we outline the algorithm to compute hinges for biconnected graphs.

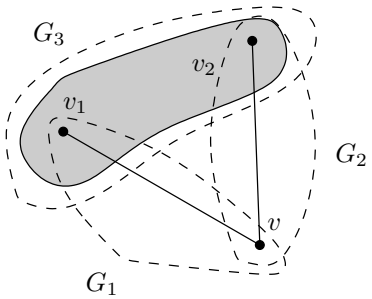

Figure 26: Decomposition based in a 2-degree vertex.

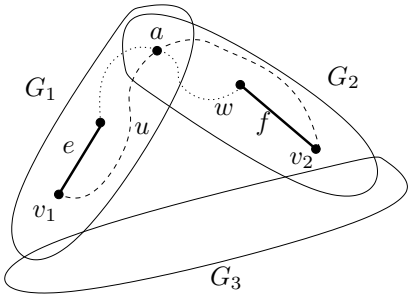

Figure 27: Paths in a circuit connecting v_1 and v_2.

5.1 Hinges and Fundamental Circuits

To establish the relation between hinges and fundamental circuits in biconnected graphs, we prove that if a graph G is decomposable, there is always a fundamental circuit C of G with nodes that are hinges. When needed, the edges of a circuit C will be denoted as $E(C)$. We start with the following lemma.

Lemma 39 *Let $G = (V, E)$ be a graph, let $\{G_1, G_2, G_3\}$ be a set decomposition of G and let C be a circuit of G. Then, one of the two following cases holds:*

1. *G_1, G_2 and G_3 contain edges of C, or*
2. *The edges of C belong to just G_1 or to G_2 or to G_3.*

Proof
Let $G = (V, E)$ be a graph, $\{G_1, G_2, G_3\}$ a set decomposition of G, and let $a \in V$ be the hinge of the set decomposition such that $\{a\} = V(G_1) \cap V(G_2)$. For a contradiction, assume that C is a circuit of G such that some edges belong to $E(G_1)$, some to $E(G_2)$ but no edges of C belong to $E(G_3)$. Let e, f be two edges in $E(C)$ such that $e \in E(G_1)$ and $f \in E(G_2)$. Since C is a circuit, there are two paths in C, say u, w, that connect e to f. See Figure 27. Moreover, since C and G_3 are disjoint, paths u and w are also disjoint with G_3. Hinge a must belong to any path connecting a vertex in G_1 to a vertex in G_2 hence $a \in u$ and $a \in w$. Therefore C is not a circuit.

□

Example 40 *Consider the circuit $C = [d, a, h]$ in the graph of Figure 28 left. Figure 28 middle shows a graph decomposition $\{G_1, G_2, G_3\}$ such that $C \subset G_3$.*

◊

Example 41 *Consider the circuit $C = [b, d, h, a, f, g, c]$ in the graph of Figure 28 left. Figure 28 right shows a graph decomposition $\{G_1, G_2, G_3\}$ such that the edges of C are spread all over the three clusters G_1, G_2 and G_3.* ◊

Next we present the main result in this work.

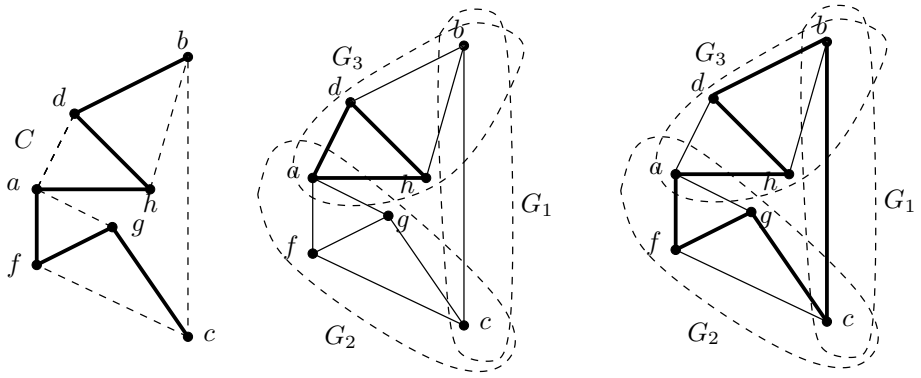

Figure 28: Left) Graph. Middle) Circuit within just one decomposition cluster. Right) Circuit spread over three decomposition clusters.

Theorem 42 *Let $G = (V, E)$ be a biconnected graph with hinges $\{a, b, c\}$ and let T be a spanning tree to G. Then there is a fundamental circuit C of G with respect to T such that the hinges $\{a, b, c\}$ belong to C.*

Proof
Let $G = (V, E)$ be a decomposable biconnected graph. Let $\{C_1, \ldots, C_m\}$ be the set of fundamental circuits of G with respect to the spanning tree T and let $\{a, b, c\} \subseteq V$ be the hinges that split G into clusters $\{G_1, G_2, G_3\}$.

For a contradiction assume that there is no fundamental circuit in $\{C_1, \ldots, C_m\}$ such that its edges span the three clusters G_1, G_2, G_3. According to Lemma 39, we have that for each fundamental circuit C, either $C \subseteq G_1$ or $C \subseteq G_2$ or $C \subseteq G_3$.

Since G is biconnected there are two vertex disjoint paths that connect any pair of given vertices, thus there is a circuit, say C', such that connects the three hinges a, b, c. Hence some edges of C' belong to G_1 some to G_2 and some to G_3 and by hypothesis C' is not a fundamental circuit and, according to what has been said in Section 2.4, can be expressed as the ring sum of some fundamental circuits of G.

But the edges of each fundamental circuit in G belong to either G_1, G_2 or G_3 and the ring sum of circuits in G_i yields a circuit just in G_i. Thus C' cannot be built as the ring sum of fundamental circuits of G. This contradiction refutes the initial hypothesis.

□

5.2 The Algorithm

If $G = (V, E)$ is a biconnected graph, decomposing it into clusters components amounts to computing a set of hinges in G. Since we have proved that hinges, if any, belong to fundamental circuits C of G, the problem of decomposing a biconnected graph can be stated as

> Given a biconnected constraint graph $G = (V, E)$, with a fundamental circuit C, compute the hinges contained in C, if any.

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

Compute an spanning tree T to G
Compute the set of fundamental circuits C of G according to T
for C_i in C **do**
    Compute the set of bridges B of G with respect to C_i
    Compute the set of merged bridges B'
    Compute the set of faces F of the planar embedding of $B' \cup C_i$
    for F_i in F **do**
        for each tern $\{v_{1j}, v_{2j}, v_{3j}\}$ in F_i **do**
            if $\{v_{1j}, v_{2j}, v_{3j}\} \in C_i$ **then return** $\{v_{1j}, v_{2j}, v_{3j}\}$ **endif**
        endfor
    endfor
endfor
return $\emptyset$

Figure 29: Decomposition of a biconnected graph.

Theorem 42 is the basis to efficiently solve this problem. The algorithm starts by computing a spanning tree using a depth-first search and, from it, computing the set of fundamental circuits of G.

Then the algorithm seeks for a fundamental circuit C with a set of hinges. Our efficient algorithm for computing the hinges is based on the decomposition of a graph in the bridges induced by C, and is inspired in the algorithm for finding the triconnected components of a graph reported by Miller and Ramachandran, [13]. If the algorithm fails finding a fundamental circuit with a set of hinges, the input graph is not decomposable. Figure 29 shows this algorithm.

6 Case Study

In this section we show how our algorithm works by following a case study. Assume that the given graph $G = (V, E)$ is the one shown in Figure 30.

6.1 Computing the Set of Fundamental Circuits

First the algorithm computes a spanning tree to the graph G by applying a DFS technique as explained in Section 2.5. If the starting arbitrary root is vertex d, Figure 31a shows the resulting spanning tree.

Then fundamental circuits are computed. Notice that every chord in the spanning tree induces a fundamental circuit as depicted in Figure 31b.

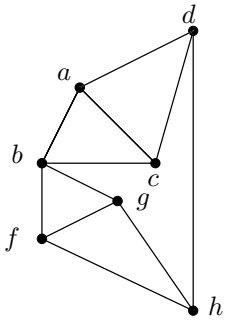

Figure 30: Case study graph.

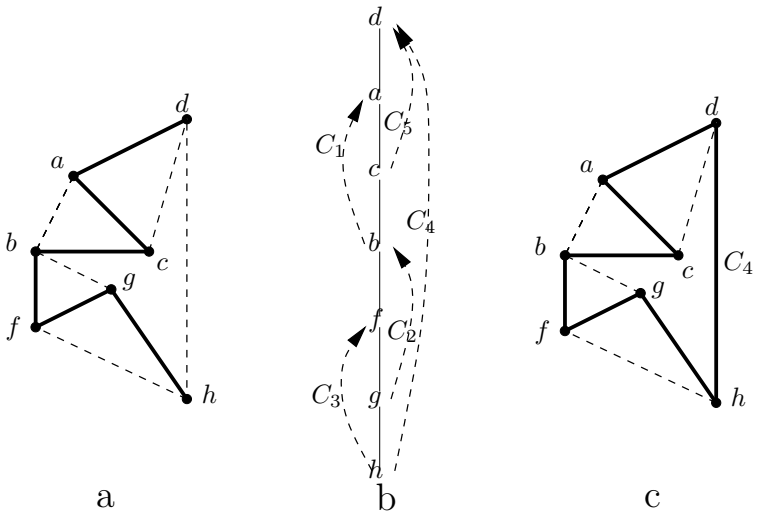

Figure 31: a) Spanning tree. b) Associated circuits. c) Chosen fundamental circuit.

$$
\begin{aligned}
C_1 &= [a, c, b] \\
C_2 &= [b, f, g] \\
C_3 &= [f, g, h] \\
C_4 &= [d, a, c, b, f, g, h] \\
C_5 &= [d, a, c]
\end{aligned}
$$

6.2 Computing the Set of Bridges

Choose an arbitrary fundamental circuit, for example $C_4 = [d, a, c, b, f, g, h]$ in Figure 31c. According to the definition given in Section 2.7, C_4 induces in G the set of components, say

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

Thus, H_1, H_2, H_3 and H_4 are also the bridges of G induced by C_4. Notice that, in this case, all the bridges are singular.

6.3 Computing the Collapsed Graph

Star graphs that replace bridges are computed as explained in Section 2.9. They are

$$
\begin{aligned}
S_1 &= \{x_1, a, b\} \\
S_2 &= \{x_2, c, d\} \\
S_3 &= \{x_3, b, g\} \\
S_4 &= \{x_4, f, h\}
\end{aligned}
$$

The collapsed graph resulting by replacing every bridge H_i by the star graph S_i is shown in Figure 32a.

6.4 Computing the Merged Graph

Stars in the collapsed graph are merged as explained in Section 2.10. Stars S_1 and S_2 have interlaced attachments and are merged into S_{12}. Similarly, stars S_3 and S_4 interlace and are merged into S_{34}. The resulting merged graph is shown in Figure 32b.

6.5 Computing the Planar Embedding

Next step is to compute a planar graph as explained in Section 2.6. Figure 33 shows the planar graph resulting from embedding the merged graph given in Figure 32b.

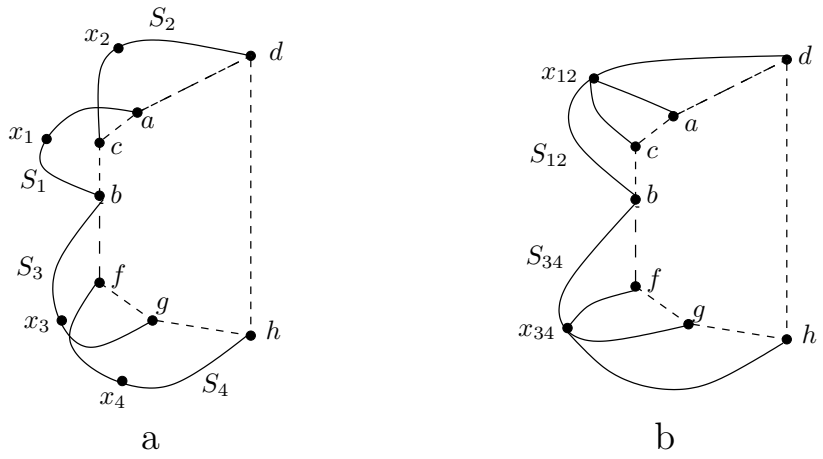

Figure 32: a) Collapsed graph. b) Merged graph.

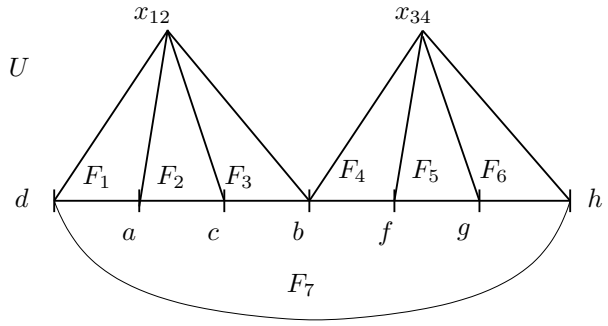

Figure 33: Planar embedding for the merged graph given in Figure 32b.

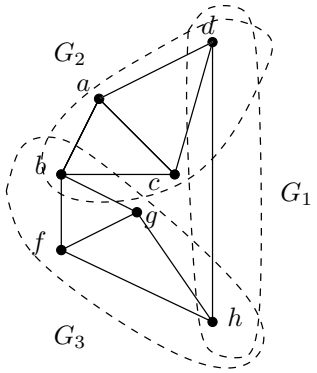

Figure 34: Graph and clusters G_1, G_2 and G_3.

The set of faces defined by the planar embedding of the merged graph is $\{U, F_1, F_2, F_3, F_4, F_5, F_6\}$. The boundaries of the faces in the planar embedding are the circuits

$$\begin{aligned} F_1 &= [d, x_{12}, a] \\ F_2 &= [a, x_{12}, c] \\ F_3 &= [c, x_{12}, b] \\ F_4 &= [b, x_{34}, f] \\ F_5 &= [f, x_{34}, g] \\ F_6 &= [g, x_{34}, h] \\ F_7 &= [d, a, c, b, f, g, h] \\ U &= [d, x_{12}, b, x_{34}, h] \end{aligned}$$

Notice that, in the planar embedding, face U is the unbounded part of the drawing.

6.6 Computing the Hinges

To compute a set of hinges, we search in the set of faces $\{F_1, F_2, F_3, F_4, F_5, F_6, F_7, U\}$.

We found that vertices $\{d, b, h\}$ in the boundary of the face labeled U also belong to the chosen circuit C_4 used to compute the planar embedding. Therefore, $\{d, b, h\}$ are hinges which split the graph G into three clusters G_1, G_2 and G_3. This completes the decomposition process. Resulting clusters are shown in Figure 34.

7 Summary and Future Work

One of the main issues in graph-based geometric constraint solving is decomposing a geometric constraint graph. In this work we have presented a new algorithm to decompose geometric constraint graphs. The algorithm is inspired in a technique reported by Miller and Ramachandran to split a graph into triconnected components, [13].

The algorithm decomposes the graph by computing terns of vertices, called hinges, which allow to split the graph into three clusters. First the algorithm transforms the constraint

graph into a planar embedding such that preserves the hinges. Then hinges are computed by searching in the fundamental circuits.

Some open problems deserve more work. We plan to study in the near future the following issues

- Correctness. In this work have pointed out some aspects about algorithm correctness. However a complete proof of the correctness must be developed. The correcteness of the planar embedding seems to be the most challenging.
- Complexity. Preliminary experiments suggest that the algorithm running time is at most quadratic with the number of nodes in the graph. Studies to accept or reject this conjecture must be conducted. It would be interesting how our algorithm compares to those reported by Fudos, [4], and by Owen,[14] .
- Improving the runnung time. When decomposing a graph, computing hinges is applied recursively. We believe that some of the intermediate results obtained after computing the hinges at a given level can be reused in the following level.

8 Acknowledgments

This research has been partially funded by Ministerio de Educacion y Ciencia and by FEDER under grant TIN2004-06326-C03-01.

References

[1] Condoor, S. S., and Condoor, S. *Mechanical Design Modeling Using Pro/Engineer.* McGraw-Hill Higher Education, 2001.

[2] Durand, C. B. *Symbolic and numerical techniques for constraint solving.* PhD thesis, Computer science department, Purdue University, 1998. Major Professor-Christoph M. Hoffmann.

[3] Even, S. *Graph Algorithms.* Computer Science Press, Potomac, Md., 1979.

[4] Fudos, I., and Hoffmann, C. M. A graph-constructive approach to solving systems of geometric constraints. *ACM Trans. Graph. 16*, 2 (1997), 179–216.

[5] Hoffmann, C., Lomonosov, A., and Sitharam, M. Decomposition plans for geometric constraint systems, part i: Performance measures for cad. *Journal of Symbolic Computation* (2001).

[6] Hoffmann, C., Lomonosov, A., and Sitharam, M. Decomposition plans for geometric constraint systems, part ii: New algorithms. *Journal of Symbolic Computation* (2001).

[7] Hoffmann, C. M., and Joan-Arinyo, R. A brief on constraint solving. *Computer-Aided Design and Applications* (2005).

[8] Hopcroft, J., and Tarjan, R. Algorithm 447: efficient algorithms for graph manipulation. *Commun. ACM 16*, 6 (1973), 372–378.

[9] Joan-Arinyo, R., Soto, A., and Vila, S. Resolución de restricciones geométricas. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial 20* (2003), 121–136.

[10] Joan-Arinyo, R., Soto-Riera, A., Vila-Marta, S., and Vilaplana, J. Declarative characterization of a general architecture for constructive geometric constraint solvers. In *The Fifth International Conference on Computer Graphics and Artificial Intelligence* (Limoges, France, 14-15 May 2002), D. Plemenos, Ed., Université de Limoges, pp. 63–76.

[11] Joan-Arinyo, R., Soto-Riera, A., Vila-Marta, S., and Vilaplana-Pastó, J. Revisiting decomposition analysis of geometric constraint graphs. In *SMA'02: Proceedings of the seventh ACM symposium on Solid modeling and applications* (New York, NY, USA, 2002), ACM Press, pp. 105–115.

[12] Joan-Arinyo, R., Soto-Riera, A., Vila-Marta, S., and Vilaplana-Pastó, J. Transforming an under-constrained geometric constraint problem into a well-constrained one. In *SM'03: Proceedings of the eighth ACM symposium on Solid modeling and applications* (New York, NY, USA, 2003), ACM Press, pp. 33–44.

[13] Miller, G. L., and Ramachandran, V. A new graph triconnectivity algorithm and its parallelization. *Combinatorica 12*, 1 (1992), 53–76.

[14] Owen, J. Algebraic solution for geometry from dimensional constraints. In *SMA'91: Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications* (New York, NY, USA, 1991), ACM Press, pp. 397–407.

[15] Soto-Riera, A. *Geometric Constraint Solving in 2D.* PhD thesis, Departament de Llenguatges i Sistemes Informàtics, UPC, 1998.

[16] Tarjan, R. E. Depth-first search and linear graph algorithms. *SIAM J. Comput. 1*, 2 (1972), 146–160.

[17] Thulasiraman, K., and Swamy, N. *Graphs: Theory and Algorithms.* John Wiley & Sons, 1992.

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

[19] Vila-Marta, S. *Contribution to Geometric Constraint Solving in Cooperative Engineering.* PhD thesis, Departament de Llenguatges i Sistemes Informàtics, UPC, 2003.

[20] Whitney, H. Non-separable and planar graphs. *Proceedings of the National Academy of Sciences of the United States of America 17*, 2 (February 1931), 125–127.