h-graphs: A New Representation for Tree Decompositions of Graphs

Marta R. Hidalgo
Systems Genomics Lab
Centro de Investigación Príncipe Felipe
València, Spain
Robert Joan-Arinyo
Grup d'Informàtica a l'Enginyeria
Universitat Politècnica de Catalunya
Barcelona, Catalunya

September 8, 2015

Abstract

In geometric constraint solving, well constrained geometric problems can be abstracted as Laman graphs. If the graph is tree decomposable, the constraint-based geometric problem can be solved by a Decomposition-Recombination planner based solver. In general decomposition and recombination steps can be completed only when other steps have already been completed. This fact naturally defines a hierarchy in the decomposition-recombination steps that traditional tree decomposition representations do not capture explicitly.

In this work we introduce h-graphs, a new representation for decompositions of tree decomposable Laman graphs, which captures dependence relations between different tree decomposition steps. We show how h-graphs help in efficiently computing parameter ranges for which solution instances to well constrained, tree decomposable geometric constraint problems with one degree of freedom can actually be constructed.

1 Introduction

Many applications in computer-aided design, computer-aided manufacturing, kinematics, robotics or dynamic geometry are conveniently modeled by geometric problems defined by geometric constraints with parameters, some of them representing dimensions. These generic models allow the user

to easily generate specific instances for various parameter and constraint values.

When parametric models are used in real applications, it is often found that instantiation may fail for some parameter values. Assuming that failures are not due to bugs in the system, they should be attributed to a more basic problem, that is, a certain combination of constraints in the model and values of parameters do not define a valid shape.

The failure to instantiate the model poses naturally the question of how to compute ranges for parameters such that model instantiation is feasible. This problem or restricted versions of it have been addressed in the literature. Shapiro and Vossler, [21], and Raghothama and Shapiro, [18, 19, 20], developed a theory on validity of parametric family of solids by investigating the relationship between Brep and CSG schemas in systems with dual representations for solid modeling. The formulation is built on formalisms of algebraic topology. Unfortunately, it seems a rather difficult problem transforming these formalisms into effective algorithms.

Joan-Arinyo and Mata [13] reported on a method to compute feasible ranges for parameters in geometric constraint solving under the assumption that values assigned to parameters are non-trivial-width intervals. The method applies to complex systems of geometric constraints in both 2D and 3D and has been successfully applied in the dynamic geometry field, [2]. It is a general method, the main drawback, however, is that it is based on numerical sampling.

Hoffmann and Kim [9] developed a constructive approach to calculate parameter ranges for systems of geometric constraints that include sets of isothetic line segments and distance constraints between them. Model instantiation for distance parameters within the ranges output by the method preserve the topology of the set of isothetic lines.

For the first time, van Der Meiden and Bronsvoort [23] described a method to directly figure out the allowable range for a single parameter in the problem, called *variant parameter*, such that an actual solution exists for any value in the range. The method was formalized by Hidalgo and Joan-Arinyo in [5, 6] where a correctness proof along with specific implementation details were given. This approach heavily relies on identifying the set of construction steps in the solution to the constraint problem the actual construction of which depend on the current value assigned to the variant parameter. So far dependences were computed on demand hence devising a method able to efficiently identify these dependences would be a valuable accomplishment.

In this work we introduce h-graphs, a new representation for graph based constructive solutions to the geometric constraint problem. The h-graph captures the relations between construction steps that dependences natu-

rally define in the description of the solution to the geometric constraint problem. Computing parameter ranges where the solution is feasible using h-graphs improves over the method described by Hidalgo and Joan-Arinyo in [5, 6].

In what follows we first give a short intuitive introduction to the geometric constraint solving problem to motivate the need for computing parameter ranges. Then we recall the graph tree decomposition, we give technical definitions related to dependence, define h-graphs, show some properties of h-graphs and, we describe an algorithm to compute h-graphs from a tree decomposition which is a solution to a geometric constraint problem. Finally we illustrate how actual dependences are computed applying the h-graph to a geometric constraint problem with one degree of freedom.

2 Geometric Constraint-Based Problems

Assume that we want to build a triangle the vertices of which are the points a, b, c like those shown in Figure 1a. We want point a to be placed at a distance d_1 from point b and point c to be placed at a distance d_2 from point b. Moreover we want that the edge bounded by points a, b makes an angle λ with respect to the edge bounded by points a, c. It is well known that this description properly defines a triangle in the Euclidean space. A ruler-and-compass procedure to build the triangle is illustrated in Figure 1b and can be described as follows,

1. Draw an arbitrary straight line, say X.
2. On line X mark an arbitrary point a.
3. On line X mark a point b at a distance d_1 from a.
4. Draw a line L through point a and at an angle λ with line X.
5. Draw a circle C with center b and radius λ.
6. Intersections of circle C and line L yield points c and c' that along with points a and b define triangles which fulfill the requirements described.

If the procedure is applied after assigning specific values to d_1, d_2 and λ, the geometric construction can be carried out depending on the specific assignment of values.

Many techniques have been reported in the literature that provide powerful and efficient methods for solving geometric problems defined by constraints. For a review, see Hoffmann *et al.*, [8]. Computer programs that solve geometric problems defined by constraints are called *solvers*. Among all the

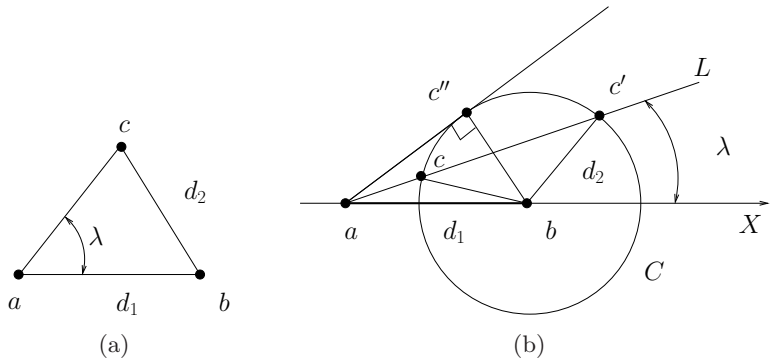

Figure 1: a) Triangle defined by three points, two point-point distances and an angle between two straight lines. b) A ruler-and-compass constructive solution.

geometric constraint solving techniques, our interest here focuses on the one known as *constructive*. See [10, 11, 12, 14, 15] and the references there in for an in depth discussion on this topic.

Constructive solvers belong to the Decomposition-Recombination solvers, in short DR-solvers, class [11] and have two main components: the *analyzer* and the *constructor*. Given the geometric elements and the constraints defined on them, the analyzer figures out a description of how geometric elements are placed with respect to each other in such a way that the constraints are fulfilled. This description is called *construction plan*.

If the analyzer succeeds, actual values are assigned to the parameters and the constructor builds an instance of a placement for the geometric objects, provided that no numerical incompatibility arises due to geometric degeneracy.

In the example described above and illustrated in Figure 1, the set of geometric elements includes the points $\{a, b, c\}$ while the constraints are the distances d_1, d_2 and the angle λ.

In this scenario asking for the set of values of λ for which the construction is actually feasible seems natural. To answer this question, efficiently computing dependences between construction steps in the construction plan plays a central role.

In what follows we only consider well constrained geometric constraint problems, that is, problems with a finite number of solution instances. In this work, these problems are abstracted as Laman graphs, [17], $G = (V, E)$ with $|V| \geq 3$ and such that

1. $|E| = 2|V| - 3$

2. For every subgraph $G' = (V', E')$ with $V' \subset V$ and $E' \subset E$, $|E'| = 2|V'| - 3$

3 Tree Decomposition of a Graph

Tree decompositions, also known as *triangular decompositions*, are a tool widely used in geometric constraint solving mainly when the underlying solving technique belongs to the DR solvers class. The resulting decomposition describes the solution to the geometric constraint problem by fixing how geometric elements are placed with respect to each other to fulfill the constraints. In this section we recall the concept of tree decomposition of a graph, we formalize the tree decomposition as a rewrite system and show some properties which will be used later on.

3.1 The Tree Decomposition

We are given a graph $G = (V, E)$ where V is a finite set of nodes or vertices and E is a collection of edges. An edge is an unordered pair (u, v) of distinct vertices $u, v \in V(G)$. In general $V(G)$ and $E(G)$ will denote respectively the set of vertices and edges of the graph G.

We start by introducing the concept of *set decomposition* illustrated in Figure 2. Let S be a set with, at least, three different members, say a, b, c. We say that three subsets of S, say S_1, S_2 and S_3 is a *set decomposition* of S if

1. $S_1 \cup S_2 \cup S_3 = S$,
2. $S_1 \cap S_2 = \{a\}$,
3. $S_2 \cap S_3 = \{b\}$ and
4. $S_3 \cap S_1 = \{c\}$.

Pairwise shared vertices a, b and c are called *hinges* for the graph set decomposition. We call the set $\{a, b, c\}$ the *hinges triplet* or just *triplet*.

Next we introduce the concept of *tree decomposition step* of a graph. See Figure 3. Let $G = (V, E)$ be a graph, the subsets $V_1(G), V_2(G)$ and $V_3(G)$ define a tree decomposition step of G if they are a set decomposition of $V(G)$ and for every edge $e = (v_1, v_2)$ with $e \in E(G)$, $v_1, v_2 \in V(G_i)$ for some i, $1 \leq i \leq 3$. Subgraphs $G_i = (V_i, E_i)$ induced in G by a tree decomposition step are called *clusters*.

Roughly speaking, a tree decomposition step of a graph G, is a graph decomposition induced by a set decomposition of vertices $V(G)$ such that the two vertices bounding each edge in $E(G)$ belong to a given cluster.

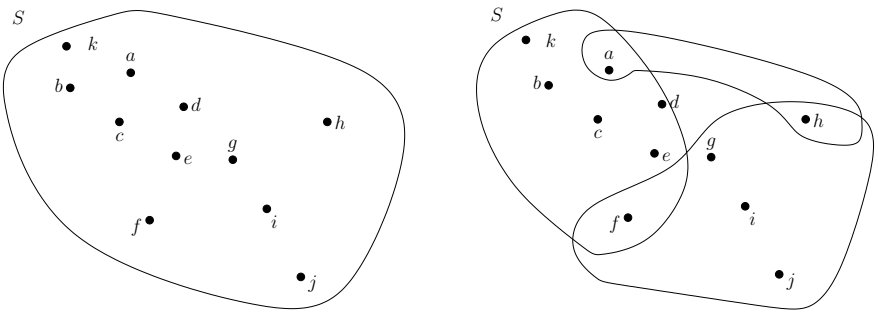

Figure 2: a) Set S with three or more members. b) Set decomposition of S.

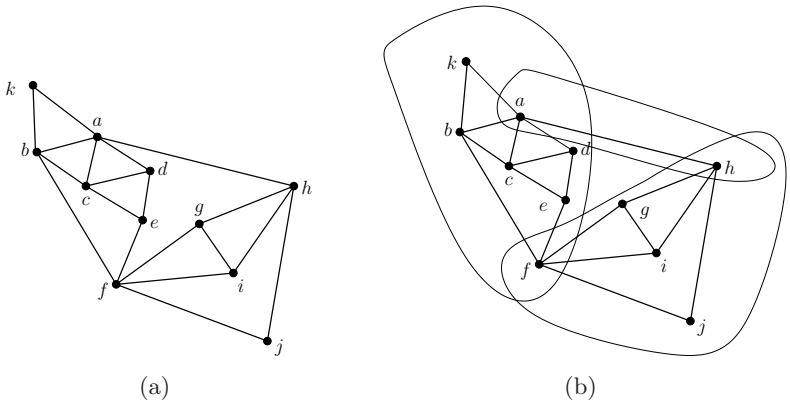

Figure 3: a) Graph. b) Graph tree decomposition step.

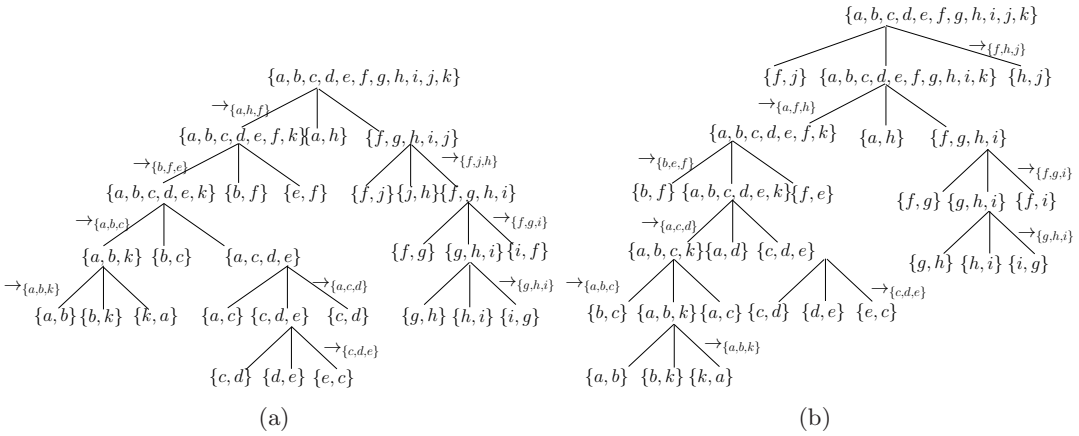

Figure 4: Two different tree decompositions for the graph shown in Figure 3a.

Finally we define the *tree decomposition* of a graph. Let $G = (V, E)$ be a graph. We say that a ternary tree T is a *tree decomposition* of G if

1. G is the root of T,
2. Each node $G' \subset G$ of T is the father of exactly three nodes, say G'_1, G'_2 and G'_3, which are the clusters output by a tree decomposition step applied to a subgraph of G, and
3. Each leaf node contains a cluster with exactly two vertices a, b of V such that edge (a, b) is in $E(G)$.

A graph for which there is a tree decomposition is called *tree-decomposable*. In general, a tree decomposition of a graph is not unique. Figure 4 shows two different tree decompositions for the graph given in Figure 3a. For the sake of clarity, tree decompositions only show the set of vertices included by clusters. Labels on the tree edges will be defined later on.

3.2 The Solution Constructor as a Rewrite System

As shown in [3, 14], the process of actually building a solution to a geometric constraint problem described as a tree decomposition of a graph, that is, the solver constructor, can be abstracted as a rewrite system, [16], where terms are sets of clusters. Given a graph $G = (V, E)$ the starting set of clusters is defined as

$$\mathbf{C}_G = \{\{u, v\} : (u, v) \in E(G)\}$$

Clusters are rewritten using the tree decomposition step as a reduction rule, which is denoted by an arrow $\rightarrow$ and formally defined as follows.

Definition 3.1. *Let* $\mathbf{C}$ *be a set of clusters where there are three clusters* $C_i, 1 \leq i \leq 3$ *such that pairwise share one vertex* $V(C_1) \cap V(C_2) = \{a\}, V(C_2) \cap V(C_3) = \{b\}, V(C_3) \cap V(C_1) = \{c\}$ *with* a, b *and* c *distinct. Then* $\mathbf{C} \to \mathbf{C}^*$ *where*

$$\mathbf{C}^* = (\mathbf{C} - \{C_1, C_2, C_3\}) \cup \{C_1 \cup C_2 \cup C_3\}$$

Definition 3.2. *A* derivation *is a sequence of applications of the rewriting rule in* $(\mathbf{C}, \to)$*. We will denote a derivation by*

$$\mathbf{C} \to^* \mathbf{C}^*$$

Definition 3.3. *A term* $\mathbf{C}^*$ *is derived from* $\mathbf{C}$ *if and only if there is a derivation such that* $\mathbf{C} \to^* \mathbf{C}^*$*. A term* $\mathbf{C}$ *to which the tree decomposition rule does not apply is called* irreducible *or* normal form*.*

To us, the most important result in reduction systems is that the reduction system $(\mathbf{C}_G, \to)$ has a unique normal form that is obtained after finitely many reductions, [3, 14]. In these conditions, if the geometric constraint problem is well constrained, that is, the problem has a finite number of solution instances, the derivation reduces the initial set $\mathbf{C}_G$ to a single cluster. The sequence of construction steps identified by the derivation places a fixed set of triplets of geometric elements in relative positions such that the constraints hold.

If $\{a, b, c\}$ are the nodes pairwise shared by clusters C_1, C_2, C_3, denote by $\to_{\{a,b,c\}}$ the reduction that merges the clusters. In these conditions, the set

$$\mathbf{R}_{\mathbf{C}}(*) = \{\{a, b, c\} \ : \to_{\{a,b,c\}} \ \in \ \to^*\}$$

is the set of triplets or reductions in the derivation $\to^*$.

According to [3] and [7], each triplet of hinges is used once and only once in a reduction process. This means that given two different reduction sequences over the same starting and ending terms, $\mathbf{C} \to^* \mathbf{C}^*$ and $\mathbf{C} \to^{*'} \mathbf{C}^*$ we have

$$\mathbf{R}_{\mathbf{C}}(*) = \mathbf{R}_{\mathbf{C}}(*')$$

Edges in the tree decompositions shown in Figure 4 are labeled with the reduction that merges three clusters into a new one. The set of triplets is the same in both tree decompositions,

$$\mathbf{R}_{\mathbf{C}}(*) = \{\{a, b, c\}, \{a, d, c\}, \{a, h, f\}, \{b, f, e\}, \{c, d, e\}, \{f, i, g\}, \{f, i, j\}, \{g, h, i\}\}$$

From now on, given a tree-decomposable graph $G = (V, E)$, the derivation $\mathbf{C}_G \to^* \mathbf{C}^*$ applied by the solver constructor to the tree decomposition of G to actually build a solution, will be called the *derivation associated* with the graph G. We close this section showing a useful property of clusters generated by a derivation.

Theorem 3.1. *Consider the derivation* $\mathbf{C} \rightarrow^* \mathbf{C}^*$. *For each pair of clusters* $C_i^*, C_j^* \in C^*$ *with* $i \neq j$, $|V(C_i^*) \cap V(C_j^*)| \leq 1$.

Proof. By definition, a cluster places a set of vertices with respect to a local framework of reference in such a way that constraints defined on them hold. For a contradiction assume that C_i^* and C_j^* are two different clusters in C^* with $u_i, v_i \in V(C_i^*)$ and $u_j, v_j \in V(C_j^*)$ such that the pairs u_i, u_j and v_i, v_j respectively designate the same pair of vertices say $u, v \in V(\mathbf{C}^*)$. Then the pairs u_i, v_i and u_j, v_j define a rigid transformation that places vertices in one cluster with respect to the other one. That is $V(C_i^*)$ and $V(C_j^*)$ belong to the same cluster. □

In what follows the clusters merged by the reduction $\rightarrow_{\{u,v,w\}}$ will be sometimes distinguished from each other by explicitly giving the pair of vertices in the reduction triplet included in the cluster, that is denoting them as $\mathbf{C}_{uv}, \mathbf{C}_{vw}$ and $\mathbf{C}_{wu}$.

4 Dependences in Tree Decompositions

We have seen in Section 3 that given a tree decomposable graph $G = (V, E)$, in general, the tree decomposition is not unique. But the set of hinges is unique and the final cluster derived by the reduction $\mathbf{C}_G \rightarrow^* \mathbf{C}$ is canonical. This means that when in a tree decomposition several reduction steps can be applied, the specific reduction selected does not matter. For example in the tree decomposition shown in Figure 4a, if the current cluster term includes the clusters $\{a, b\}, \{b, c\}, \{c, a\}, \{f, g\}, \{g, i\}, \{i, f\}$, then reductions $\{a, b\}, \{b, c\}, \{c, a\} \rightarrow_{\{a,b,c\}} \{a, b, c\}$ and $\{f, g\}, \{g, i\}, \{i, f\} \rightarrow_{\{f,g,i\}} \{f, g, i\}$ can be applied in any sequence without affecting the resulting set of clusters.

However, in general, some reductions in a tree decomposition derivation can only be carried out after completing some other reductions. For example, reduction $\rightarrow_{\{f,h,j\}}$ in Figure 4a can be completed only after completing reductions $\rightarrow_{\{f,g,i\}}$ and $\rightarrow_{\{g,h,i\}}$. We say that reduction $\rightarrow_{\{f,h,j\}}$ *depends* on reductions $\rightarrow_{\{f,g,i\}}$ and $\rightarrow_{\{g,h,i\}}$. Dependence naturally introduces a hierarchy in the reduction steps of a derivation over a tree decomposition.

We start by defining the concept of minimal well constrained cluster induced by two vertices within a cluster.

Definition 4.1. *Consider a cluster* $\mathbf{C}_i$ *in the current term* $\mathbf{C}$. *Let* u, v *be two vertices in* $V(\mathbf{C}_i)$. *Then* $mwc_{\mathbf{C}_i}(u, v)$ *is the* minimal well constrained cluster *in* $\mathbf{C}_i$ *such that* u, v *are in* $V(mwc_{\mathbf{C}_i}(u, v))$.

Clusters in $(\mathbf{C}_G, \rightarrow)$ are well constrained, tree-decomposable graphs. Moreover the set of reductions $\mathbf{R}_C(*)$ is unique. Therefore given a cluster $\mathbf{C_i}$

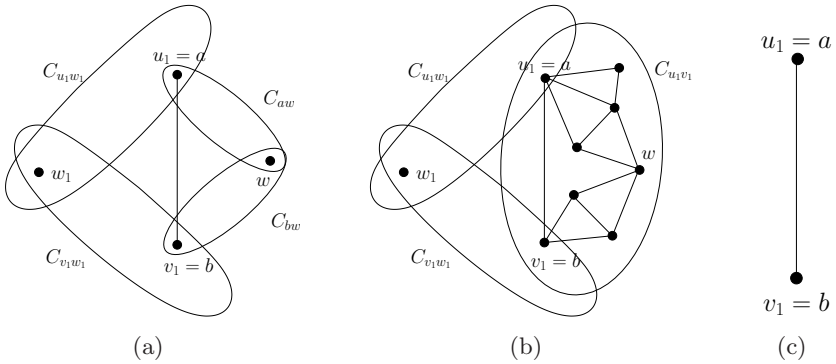

Figure 5: Direct dependence of reduction $\rightarrow_{\{u_1,v_1,w_1\}}$. a) General case. b) Particular example. c) The $mwc_{\mathbf{C}_{u_1v_1}}$ cluster in the particular example merged by the reduction $\rightarrow_{\{a,b,w\}}$ on which reduction $\rightarrow_{\{u_1,v_1,w_1\}}$ directly depends.

and two vertices $u, v \in V(\mathbf{C}_i)$, the $mwc_{\mathbf{C}_i}(u, v)$ is well defined and there is a derivation $\mathbf{C}_G \rightarrow^* mwc_{\mathbf{C}_i}(u, v)$ with reductions in $\mathbf{R}_C(*)$.

In what follows clusters $\mathbf{C_i}$ shall be denoted as $\mathbf{C_{ab}}$ where a and b are vertices in a triplet $\{u, v, w\}$ on which a reduction is applied. Thus we will denote the minimal well constrained cluster $mwc_{\mathbf{C}_{ab}}(a, b)$ just as $mwc_{\mathbf{C}_{ab}}$.

We consider two different dependence categories: *direct dependence* and *indirect dependence*. Technically we define them as follows. Refer to Figure 5.

Definition 4.2. *Consider the term* $\mathbf{C}$*. Let* $\rightarrow_{\{u_1,v_1,w_1\}}$ *be a reduction in* $\mathbf{R_C}(*)$ *involving three clusters* $\mathbf{C}_{ab}$ *with* $a, b \in \{u_1, v_1, w_1\}$ *and* $a \neq b$*. Let* $mwc_{\mathbf{C}_{ab}}$ *be such that* (a, b) *is an edge in* $E(mwc_{\mathbf{C}_{ab}})$*. Let* $\rightarrow_{\{a,b,w\}}$ *be the reduction in* $\mathbf{R_C}(*)$ *which merged the clusters* $(\{a, b\}, \{(a, b)\}), \mathbf{C}_{bw}$ *and* $\mathbf{C}_{wa}$*. Then we say that reductions* $\rightarrow_{\{u_1,v_1,w_1\}}$ *and* $\rightarrow_{\{a,b,w\}}$ *directly depend on each other.*

Direct dependence is symmetric and relates a set of reductions at the same hierarchical level. Figure 5a shows the general case of direct dependence, Figure 5b is a particular example and Figure 5c details the minimal well constrained cluster $mwc_{\mathbf{C}_{ab}}$ resulting from applying the reduction $\rightarrow_{\{a,b,w\}}$.

Indirect dependence relates reductions at different hierarchical levels by capturing the idea that there are reductions which can be carried out only after completing other reductions. Generically we define indirect dependence as follows.

Definition 4.3. *Consider the term* $\mathbf{C}$*. Let* $\rightarrow_{\{u_1,v_1,w_1\}}$ *be a reduction in* $\mathbf{R_C}(*)$ *involving three clusters* $\mathbf{C}_{ab}$ *with* $a, b \in \{u_1, v_1, w_1\}$ *and* $a \neq b$*. Let*

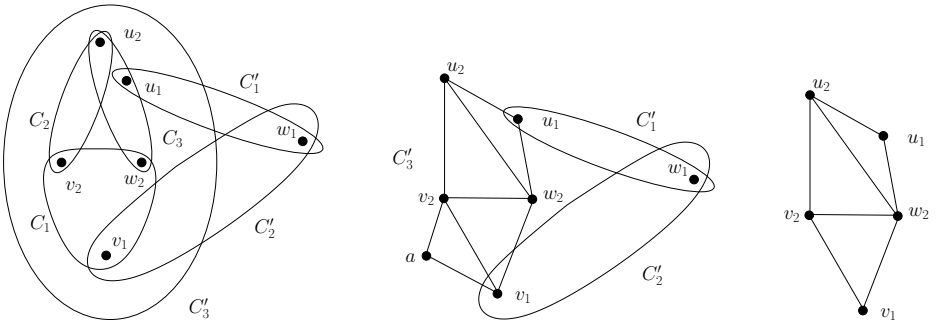

Figure 6: Indirect dependence of reduction $\rightarrow_{\{u_1,v_1,w_1\}}$. a) General case. b) Particular example. c) The $mwc_{\mathbf{C}_{u_1v_1}}$ cluster in the particular example.

$mwc_{\mathbf{C}_{ab}}$ be such that $|E(mwc_{\mathbf{C}_{ab}})| > 3$. We say that reduction $\rightarrow_{\{u_1,v_1,w_1\}}$ indirectly depends on the set of reductions in the derivation $\mathbf{C}_G \rightarrow^* mwc_{\mathbf{C}_{ab}}$.

It is easy to see that indirect dependence is transitive.

Figure 6a illustrates the definition of indirect dependence in the general case. Figure 6b is a particular example and Figure 6c details the minimal well constrained cluster $mwc_{\mathbf{C}_{u_1v_1}}$. Notice that on the one hand vertex a and the edges incident on it have been removed without affecting the rigidity of the resulting graph. On the other hand, it is clear that before attempting to apply reduction $\rightarrow_{\{u_1,v_1,w_1\}}$, reductions $\rightarrow_{\{u_1,u_2,w_2\}}, \rightarrow_{\{u_2,v_2w_2\}}$ and $\rightarrow_{\{v_1,v_2,w_2\}}$ must be completed to build the cluster $mwc_{\mathbf{C}_{u_1v_1}}$.

In these conditions, consider the reduction

$$\mathbf{C}_{uv}, \mathbf{C}_{vw}, \mathbf{C}_{wu} \rightarrow_{\{u,v,w\}} \mathbf{C}_{uvw}$$

Let $\mathbf{R}_{ab}(*)$ denote the set of reductions in the derivation $\mathbf{C}_G \rightarrow^* mwc_{\mathbf{C}_{ab}}$. Clearly reduction $\rightarrow_{\{u,v,w\}}$ indirectly depends on the reductions in the set $\mathbf{R}_{uv}(*) \cup \mathbf{R}_{vw}(*) \cup \mathbf{R}_{wu}(*)$.

5 The h-graph

Dependences between reductions over a tree decomposition of a graph are not explicitly captured by the tree representation. Here we introduce a new way to represent a tree decomposition of a graph which explicitly captures the hierarchy introduced by dependences in the reduction steps over the tree decomposition.

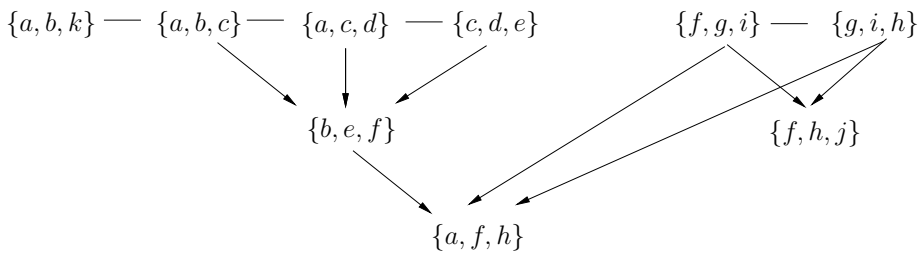

Figure 7: h-graph $\mathcal{H}_G$ associated to the tree decomposition in Figure 4a for the graph G shown in Figure 3a.

5.1 h-graph Definition

The new representation for graph tree decompositions is called *hinges graph*, in short *h-graph*, and we formally define it as follows.

Definition 5.1. *Let $G = (V, E)$ be a tree-decomposable graph. The* h-graph*, associated to G is the graph $\mathcal{H}_G = (\mathcal{V}, E_D \cup E_S)$, where $\mathcal{V}$ is the set of reductions in the associated derivation $\mathbf{C}_G \to^* \mathbf{C}^*$, E_D is the set of unordered pairs (ν_1, ν_2) such that reductions ν_1, ν_2 are directly dependent on each other and, E_S is the set of ordered pairs (ν_1, ν_2) such that reduction ν_2 indirectly depends on reduction ν_1.*

Direct dependences are represented by non-directed edges. Indirect dependencies are represented by directed edges. Figure 7 shows the h-graph $\mathcal{H}_G$ associated to the tree decompositions in Figure 4 for the graph G in turn shown in Figure 3a.

We shall now show that the h-graph for a given graph is unique.

Theorem 5.1. *Let $G = (V, E)$ be a tree-decomposable graph. The h-graph $\mathcal{H}_G$ associated with G is unique.*

Proof. Let T be a tree decomposition of G. The set of hinges triplets in T or equivalently the set of reductions in the derivation $\mathbf{C}_G \to^* \mathbf{C}^*$ is unique. Thus the set of nodes $\mathcal{V}(\mathcal{H}_G)$ is unique. Since the derivation is canonical, dependences between reductions in it are intrinsic to the graph G. □

5.2 Complete h-subgraphs

Given a tree-decomposable graph $G = (V, E)$ and the associated derivation $\mathbf{C}_G \to^* \mathbf{C}^*$ for it, there is an associated $\mathcal{H}_G$. Moreover, given a tree-decomposable subgraph $G' \subset G$ and a derivation for it $\mathbf{C}_{G'} \to^* \mathbf{C}'^*$, there is a h-graph $\mathcal{H}_{G'}$.

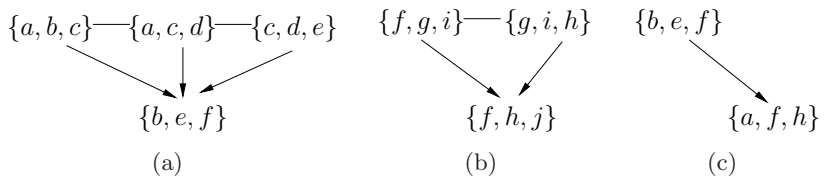

Figure 8: a), b) Complete h-subgraphs. c) An incomplete h-subgraph.

Consider now the hgraph $\mathcal{H}_G$ and let $\mathcal{H}'$ be a subgraph of $\mathcal{H}_G$ defined in the usual way. Due to the dependences, it is unclear whether there is a tree-decomposable subgraph G' of G such that $\mathcal{H}' = \mathcal{H}'_{G'}$. In what follows we describe the conditions under which G' exists. First we define h-subgraphs.

Definition 5.2. *Let $G = (V, E)$ be a tree-decomposable graph and $\mathcal{H}_G = (\mathcal{V}, E_D \cup E_S)$ the associated h-graph. $\mathcal{H}'_G = (\mathcal{V}', E'_D \cup E'_S)$ is a h-subgraph of $\mathcal{H}_G$ if and only if $\mathcal{V} \subset \mathcal{V}'$, $E'_D \subset E_D$ and $E'_I \subset E_S$.*

If ν and ν' are reductions in $\mathcal{V}(\mathcal{H}_G)$ such that ν' directly or indirectly depends on ν the set

$$\text{dep}^{\nu}_{\mathcal{H}_G} = \{\nu' : \nu' \in \mathcal{V} \text{ which depends on } \nu\}$$

is the set of reductions in $\mathcal{H}_G$ that depend on reduction ν.

Definition 5.3. *Let $G = (V, E)$ be a tree decomposable graph and $\mathcal{H}_G = (\mathcal{V}, E_D \cup E_S)$ the associated hgraph. Consider the h-subgraph $\mathcal{H}'_G = (\mathcal{V}', E'_D \cup E'_S) \subseteq \mathcal{H}_G$. We say that the h-subgraph $\mathcal{H}'_G$ is* complete *if for all $\nu \in \mathcal{V}'$, $dep^{\nu}_{\mathcal{H}_G} \subseteq \mathcal{V}'$.*

Figures 8a and 8b show complete h-subgraphs of the h-graph in Figure 7. Notice that complete h-subgraphs include all the reductions needed to complete a derivation for the h-subgraph. Figure 8c shows a h-subgraph where reduction $\rightarrow_{\{b,e,f\}}$ cannot be carried out because reductions $\rightarrow_{\{a,b,c\}}$, $\rightarrow_{\{a,c,d\}}$ and $\rightarrow_{\{c,d,e\}}$ are missing. Hence this h-subgraph is not complete.

Figure 9 shows tree decompositions built from the complete h-subgraphs in Figures 8a and 8b.

Theorem 5.2. *Let $G = (V, E)$ be a tree-decomposable graph, $\mathcal{H}_G$ the associated h-graph and $\mathcal{H}'_G$ a complete h-subgraph of $\mathcal{H}_G$. Then there is a tree decomposable subgraph $G' \subseteq G$ such that $\mathcal{H}'_{G'} = \mathcal{H}'_G$.*

Proof. If $\mathcal{H}'_G = \mathcal{H}_G$ the theorem trivially holds.

Assume now that $\mathcal{H}'_G(\mathcal{V}, E_D \cup E_S)$ is a complete proper h-subgraph of $\mathcal{H}_G$. Clearly a graph, say $G' = (V', E')$, can always be built from $\mathcal{H}'_G$ with $V' \subseteq \mathcal{V}$

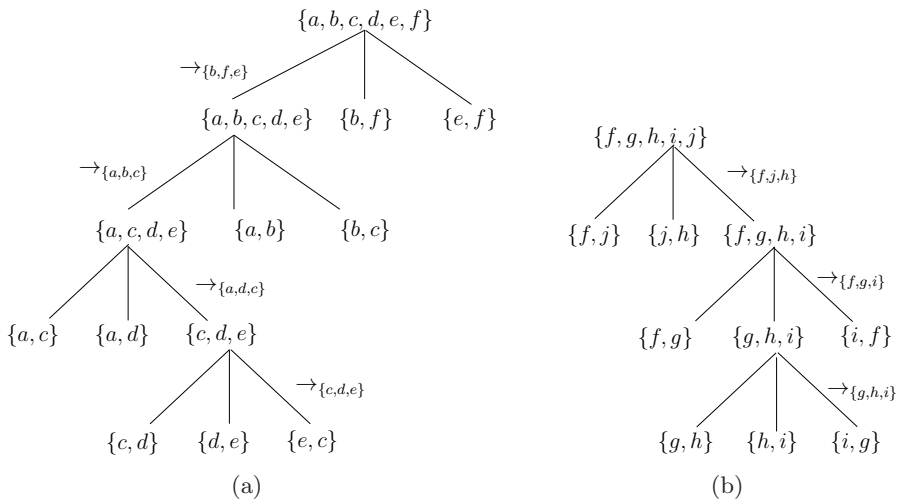

Figure 9: Tree decompositions built from complete h-subgraphs in Figures 8a and 8b.

and $E' \subseteq E_D \cup E_S$. Given that ${\mathcal{H}'}_G$ is complete, the set of decomposition steps in G' built from ${\mathcal{H}'}_G$ can be carried out thus G' is tree decomposable. □

Figure 10 shows graphs associated to the complete h-subgraphs in Figures 8a and 8b which illustrate Theorem 5.2.

Finally we prove that inclusion is preserved for h-subgraphs.

Theorem 5.3. *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two tree-decomposable graphs, and $\mathcal{H}_{G_1}, \mathcal{H}_{G_2}$ the respective associated h-graphs. Then, $\mathcal{H}_{G_1} \subset \mathcal{H}_{G_2}$ if and only if $G_1 \subset G_2$.*

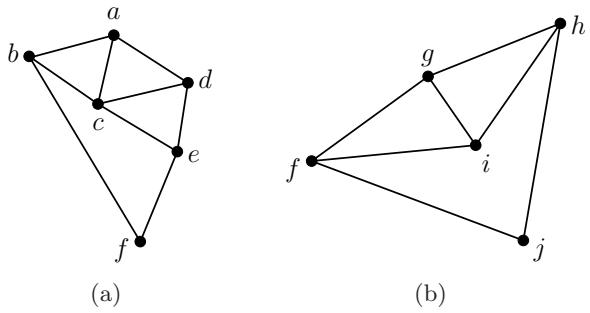

Figure 10: Tree decomposable subgraphs corresponding to the tree decompositions in Figures 9a and 9b.

Proof. For the if part apply Theorem 5.2 with $G' = G_1$ and $G = G_2$.

For the only if part assume that $\mathcal{H}_{G_2}(\mathcal{V}_2, E_{I_2} \cup E_{D_2}) \subseteq \mathcal{H}_{G_1}(\mathcal{V}_1, E_{I_1} \cup E_{D_1})$. This would imply that $V_2 \subseteq V_1$ or $E_2 \subseteq E_1$ or both hold. Therefore $G_2 \subseteq G_1$. This contradiction completes the proof. □

5.3 Computing the minimal well constrained cluster

Before describing how the minimal well constrained cluster is computed we need to prove some simple results. Let $deg(v)$ denote the degree of a vertex v in a given graph $G = (V, E)$.

Theorem 5.4. *Let $G = (V, E)$ be a well constrained graph with $|V(G)| > 3$. Then there is at least one vertex $v \in V(G)$ such that $deg(v) \geq 3$.*

Proof. Because G is well constrained we have that $|E| = 2|V| - 3$. For a contradiction assume that for each vertex $v \in V(G)$, $deg(v) = 2$. Then $|E| = |V|$. □

Theorem 5.5. *Let $G = (V, E)$ be a well constrained graph with $|V(G)| \geq 3$. The graph $G' = (V', E')$ resulting from removing a vertex of degree two from $V(G)$ and the edges in $E(G)$ incident on v is well constrained.*

Proof. By hypothesis $|E| = 2|V| - 3$. But $|V'| = |V| - 1$ and $|E'| = |E| - 2$. Then $|E'| + 2 = 2(|V'| + 1) - 3$. That is $|E'| = 2|V'| - 3$. □

Theorem 5.6. *Let $G = (V, E)$ be a well constrained graph with $|V(G)| \geq 3$. The graph $G' = (V', E')$ resulting from removing a vertex of degree three or higher from $V(G)$ and the edges in $E(G)$ incident on v is no longer well constrained.*

Proof. Let v be the vertex with $deg(v) \geq 3$ removed from $V(G)$. By hypothesis $|E| = 2|V| - 3$. But $|V'| = |V| - 1$ and $|E'| = |E| - deg(v)$. Then $|E'| + deg(v) = 2(|V'| + 1) - 3$. That is $|E'| = 2|V'| - 3 + (2 - deg(v))$. Now, $deg(v) \geq 3$ makes that $2 - deg(v) \leq -1$. Hence $|E'| < 2|V'| - 3$. □

Taking into account the results shown above, to compute the minimal well constrained cluster we need to distinguish two situations. First consider that the cluster is a leaf node of the graph tree decomposition which contains the edge (u, v). Then trivially the edge itself is the minimal well constrained cluster.

For clusters with three or more vertices we compute the minimal well constrained cluster which includes vertices u, v in two steps. First we compute the lowest common ancestor of vertices u and v, denoted $LCA_G(u, v)$, in the

tree decomposition T associated to the constraint graph at hand G. Clearly this is the smallest cluster in T which includes vertices u and v. We also identify the subset of hinges triplets of T that tree decomposes $LCA_G(u, v)$, namely $\mathcal{T}$. Second we iteratively remove from $LCA_G(u, v)$ all the two degree vertices except possibly vertices u and v as well as the corresponding triplets in $\mathcal{T}$. If the cluster is made up of three edges, the three vertices in the problem have degree two and the minimal well constrained cluster is the edge (u, v) resulting from removing the third vertex.

In the general case, the minimal well constrained cluster includes the two vertices considered u, v plus, at least, one vertex with degree three or higher. In this case, we recursively remove from the $LCA_G(u, v)$ two out of the three subclusters induced by triples $\{u^*, v^*, w^*\}$ in $\mathcal{T}$ while keeping the subcluster, say G_{ab}, such that $u, v \in V(G_{ab})$. Let $G = (V, E)$ be the constraint graph resulting from removing the degree two vertices from $LCA_G(u, v)$, let $\mathcal{T}$ denote the set of remaining hinges triplets that tree decompose G and let S denote the pair $\{u, v\}$. We handle this situation with the procedure shown in Algorithm 1 where S is stored in a static variable.

Algorithm 1 Computing the mwc in the general case

procedure mwc_general(G, $\mathcal{T}$)
  Identify a triplet $\{u^*, v^*, w^*\}$ in $\mathcal{T}$ and a cluster G_{ab} in G such that
    i) $S \subset \{u^*, v^*, w^*\}$,
    ii) $G = G_{u^*v^*} \cup G_{v^*w^*} \cup G_{w^*u^*}$, and
    iii) $S \subset V(G_{ab})$ for some $a, b \in \{u^*, v^*, w^*\}$
  if such a triplet $\{u^*, v^*, w^*\}$ exists **then**
    $\mathcal{T} = \{\{u, v, w\} \in \mathcal{T} : \{u, v, w\}$ is a triplet to $G_{ab}\}$
    mwc_general(G_{ab}, $\mathcal{T}$)
  else
    return G

As described in Section 3.2 the set of hinges triplets in a tree decomposition T is unique. Hence the set of reductions that builds the minimal tree-decomposable cluster is well defined and is easily identified by visiting nodes in the tree-decomposition T associated to the constraint graph under consideration. We only need to visit nodes starting at the T leaf nodes including vertex u or v up to the lowest common ancestor of vertices u and v.

5.4 Computing h-graphs from Tree Decompositions

Next we describe how to compute the h-graph associated to a given graph $G = (V, E)$ for which a tree decomposition T is known. The approach takes advantage of two facts. First, as described in Section 3.2, the reduction system $(\mathbf{C}_G, \rightarrow)$ is canonical. Therefore when more than one reduction

applies, the specific sequence in which they are applied does not matter and we choose to apply first those reductions which rewrite edges in the starting term $\mathbf{C}_G$ into triangles. Second, dependences between reductions over the starting term, if any, are direct dependences. Having these facts in mind the algorithm includes three main steps. First we compute an starting graph by identifying in the tree decomposition T the set of starting direct dependencies. Then a raw h-graph is computed by expanding the starting graph with the remaining direct dependences and the full set of indirect dependencies. In the last stage, the h-graph is simplified.

The algorithm makes use of the following data structures. A reduction r represents a tuple $(C_{uv}, C_{vw}, C_{wu}, u, v, w)$ where C_{uv}, C_{vw}, C_{wu} are the three clusters to be rewritten and u, v, w is the hinges triplet for the reduction.

The set of clusters in the current term of the rewriting system is stored in a list C.

The set of reductions which can be applied on the current set of clusters C is stored in the list R. This list is provided with the iterator $R.first()$ and $R.next()$.

The list V collects the set of the h-graph vertices. Each vertex v in $V = \mathcal{V}(\mathcal{H}_G)$ will store a reduction r.

The procedure to compute the starting h-graph is described in Algorithm 2. The algorithm is fed with a pointer to the tree decomposition T. The starting h-graph is the empty graph and C is initialized to the set of clusters corresponding to the leaf nodes in T. R initially stores the set of reductions which can be applied on the set of the tree decomposition leaf nodes. Clearly, for any pair of reductions in R they are either independent or directly dependent on each other. Accordingly, the set of directly dependent edges ED is conveniently updated. Finally, reductions in R are included in V and the set of clusters C is updated by serially applying the reductions in R to the current rewriting system term.

Once the h-graph has been initialized, the computation proceeds as described in Algorithm 3. In the graph-based constructive geometric constraint solving approach, the solver constructor has figured out a solution to the constraint problem when all the vertices in the graph are placed with respect to a common framework of reference. This means that all the construction steps have been carried out. Equivalently, the underlying rewrite system has just rewritten the normal form and $|C| = 1$ should hold for the number of clusters in the current term.

The loop in the algorithm first identifies one reduction which applies to the current term. Then the minimal well constrained clusters induced by the vertices in the reduction triplet within each cluster involved in the reduction cosidered are computed. Next h-graph edges are identified and included in

Algorithm 2 Initializing the h-graph

```
procedure initial_h_graph(T)
    V = ∅
    ED = ∅
    EI = ∅
    C = set of terminal nodes in T
    R = set of reductions which can be applied on C
    r1 = R.first()
    while r1 ≠ nil do
        r2 = R.next(r1)
        while r2 ≠ nil do
            if direct_dependence(r1, r2) then
                ED = ED + (r1, r2)
            r2 = R.next(r2)
        r1 = R.next(r1)
    for r in R do
        V = V + r
        C = (C - {r.Cuv, r.Cvw, r.Cwu}) ∪ {r.Cuv ∪ r.Cvw ∪ r.Cwu}
    return V, ED, EI, C
```

Algorithm 3 Computing the h-graph from a tree decomposition

```
procedure h_graph(T)
    V, ED, EI, C = initial_h_graph(T)
    while |C| > 1 do
        r = reduction which can be applied on Cuv, Cvw, Cwu
        mwcuv = mwc(T, r.Cuv, r.u, r.v)
        mwcvw = mwc(T, r.Cvw, r.v, r.w)
        mwcwu = mwc(T, r.Cwu, r.w, r.u)
        L = ∅
        for Cab in {mwcuv, mwcvw, mwcwu} do
            if Cab = ({a,b}, {(a,b)}) then
                ED = ED + (r, mwc_reduction(Cab, T))
            else
                L = L + mwc_reductions(Cab, T)
        for l in L do
            EI = EI + (r, l)
        V = V + r
        C = (C - {r.Cuv, r.Cvw, r.Cwu}) ∪ {r.Cuv ∪ r.Cvw ∪ r.Cwu}
    EI = transitive_reduction(V, EI)
    return V, ED, EI
```

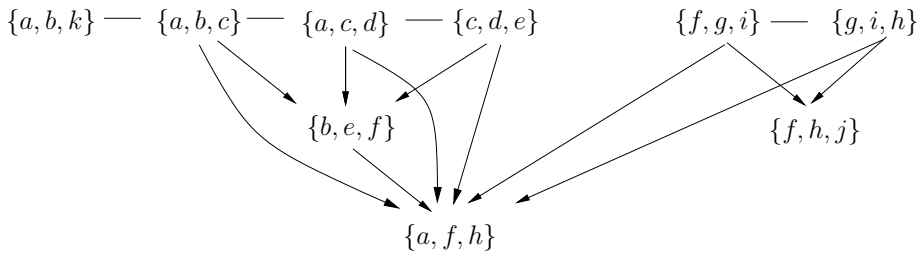

Figure 11: Raw h-graph for the tree decomposition in Figure 4a.

the corresponding set of h-graph edges according to whether the dependences are either direct, ED, or indirect, EI. Finally the reduction under consideration is both added to the set of h-graph vertices, $\mathcal{V}(\mathcal{H}_G)$, and actually applied to the current term C to update it.

When the main loop in Algorithm 3 is over, we end up with a raw h-graph. For example, for the tree decompositions shown in Figure 4, the h-graph generated would be the one depicted in Figure 11.

In general, the raw h-graph includes some extra edges. Taking into account that indirect dependence is transitive, directed edges $(\{a, b, c\}, \{a, h, f\})$, $(\{a, c, d\}, \{a, h, f\})$ and $(\{c, d, e\}, \{a, h, f\}) \in \mathcal{V}(\mathcal{H}_G)$ in the raw graph depicted in Figure 11 are redundant. If we do not consider undirected edges, the raw h-graph is a directed acyclic graph. Thus the final h-graph is computed as the unique transitive reduction of the raw h-graph, [1, 4]. The h-graph output by the simplification process is shown in Figure 7.

6 Computing Dependences in 1 DOF Problems

Consider the geometric constraint problem described in Section 2. When values for, say, distances d_1 and d_2 are fixed while the value assigned to the angle λ changes, the problem has one degree of freedom and λ is the variant parameter. This geometric constraint problem is abstracted by the graph $G = (V, E)$ in Figure 12 where edge (c, d) defines the variant parameter λ for the degree of freedom in the underlying geometric problem. The associated h-graph $\mathcal{H}_G = (\mathcal{V}, E_D \cup E_S)$ is shown in Figure 7.

Constraint-based geometric problems with one degree of freedom are in the core of, among others, parametric solid modeling and dynamic geometry. [5, 6, 23]. In these fields, knowing beforehand which is the set of values for λ such that the geometric construction can actually be built plays a central role. This challenging and longtime standing problem used to be solved by regular sampling in the parameters space. See [2].

A direct method to compute the set of values of λ for which the construction

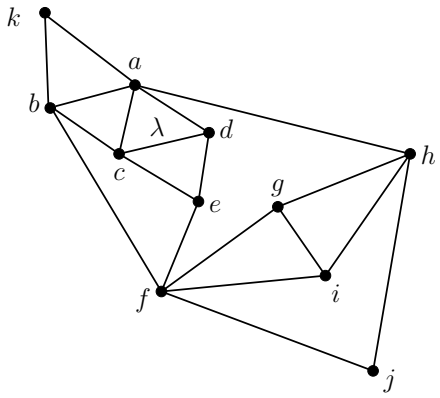

Figure 12: Graph in Figure 3a where edge (c, d) defines the variant parameter λ for the degree of freedom in the underlying geometric problem.

plan solution to a constraint problem is feasible was described for the first time in [22, 23]. The method was formalized in [5, 6] where a correctness proof along with specific implementation details were given.

The approach heavily relies on computing the set of reduction steps on which the variant parameter depends. So far, dependences are computed on demand for each specific situation found when considering each construction step while progressing in the identification of parameter ranges. Hence, devising an efficient method to identify reductions dependencies is paramount.

In this section we describe an algorithm to compute parameter dependences in geometric constraint problems with one degree of freedom. The algorithm efficiently computes the set of reduction steps over a tree decomposition of a graph $G = (V, E)$ which depend on the variant parameter λ. The hierarchy captured by the h-graph $\mathcal{H}_G = (\mathcal{V}, E_D \cup E_S)$ associated to the tree decomposition of graph G leverages the computation of dependencies and the method improves over the approach reported in [5]. Recall that given a tree-decomposable graph G, the associated $\mathcal{H}_G$ is unique.

Dependences are computed in two steps as described in Algorithm 4. Assume that the variant parameter is $\lambda = (u, v) \in E(G)$. A reduction in $V(\mathcal{H}_G)$ such that vertices u, v belong to the hinges triplet directly depend on the variant parameter λ. Therefore, the set of reductions which directly depend on λ is computed visiting once each vertex in $V(\mathcal{H}_G)$.

To figure out the set of reductions which indirectly depend on λ all what we have to do is, with the help of a stack to store reductions, recursively visit the reductions in $V(\mathcal{H}_G)$ on which the set of direct dependences already computed indirectly depend. The sets DD and ID stand for the reductions on which λ respectively depends directly and indirectly.

Algorithm 4 Computing dependencies of the variant parameter

```
procedure dependences((u,v), HG)
    DD = {r ∈ V(HG) : u,v ∈ r}
    ID = ∅
    S = ∅
    for r ∈ DD do
        if there is an edge e in ES(HG) such that e = (r, r') then
            S.push(r)
    while not S.empty() do
        r = S.pop()
        E = {e ∈ ES(HG) : e = (r, r')}
        for e ∈ E do
            ID = ID + r'
            S.push(r')
    return DD, ID
```

To illustrate how the algorithm works consider again the graph $G = (V, E)$ in Figure 12 where edge (c, d) defines the variant parameter λ and the associated h-graph $\mathcal{H}_G = (\mathcal{V}, E_D \cup E_S)$ is shown in Figure 7. Reductions in $\mathcal{V}(\mathcal{H}_G)$ which include vertices c and d as hinges are $\rightarrow_{\{a,c,d\}}$ and $\rightarrow_{\{c,d,e\}}$. These reductions directly depend on λ. Reductions which indirectly depend on $\rightarrow_{\{a,c,d\}}$ and $\rightarrow_{\{c,d,e\}}$ are $\rightarrow_{\{b,e,f\}}$ and $\rightarrow_{\{a,f,h\}}$. These reductions indirectly depend on λ.

For a detailed description on how to compute feasible ranges for the variant parameter in geometric constraint problems with one degree of freedom see [5, 7]. As an example, assume that edges $(a, c), (a, d), (c, e)$ and (e, d) are point-point distance constraints with values d_1, d_2, d_3 and d_4 respectively. Assume that $\lambda = (c, d)$ is a point-point distance constraint for which we want to know the range of values such that both reductions, $\rightarrow_{\{a,c,d\}}$ and $\rightarrow_{\{c,d,e\}}$, can be carried out. This range can now be figured out as follows. On the one hand, the triangle inequality for reduction $\rightarrow_{\{a,c,d\}}$ fixes that $|d_1 - d_2| \leq \lambda \leq d_1 + d_2$. On the other hand for reduction $\rightarrow_{\{c,d,e\}}$, we have that $|d_3 - d_4| \leq \lambda \leq d_3 + d_4$. Denote $\lambda_{min} = \max(|d_1 - d_2|, |d_3 - d_4|)$ and $\lambda_{max} = \min(d_1 + d_2, d_3 + d_4)$. Then the set of values for λ is $\lambda_{min} \leq \lambda \leq \lambda_{max}$.

7 Summary

In this work we have formalized the concept of *dependence* between construction steps in a constructive solution to a geometric constraint problem based on tree decompositions. A new representation, called h-graph, which captures both the tree decomposition and construction steps dependences

has been introduced.

Given a well constrained, tree-decomposable geometric constraint problem abstracted as a graph, the associated h-graph is unique thus there is no need to recompute it when considering different variant parameters within the constraint problem. Once the variant parameter is fixed, the h-graph allows to identify the set of construction steps which depend on it.

Identifying dependences between construction steps plays a central role in computing parameter ranges where solutions are feasible. This computation clearly benefits from h-graphs and the resulting approach outperforms those previously reported in the literature.

Acknowledgment

Robert Joan-Arinyo has been supported by the CICYT Spanish Research Agency under the project TIN2013-47137-C2-1-P.

References

[1] A.V. Aho, M.R. Garey, and J.D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

[2] M. Freixas, R. Joan-Arinyo, and A. Soto-Riera. A constraint-based dynamic geometry system. *Computer-Aided Design*, 42(2):151–161, February 2010. DOI:10.1016/j.cad.2009.02.016.

[3] I. Fudos and C.M. Hoffmann. Constraint-based parametric conics for CAD. *Computer Aided Design*, 28(2):91–100, 1996.

[4] M. Habib, M. Morvan, and J.-X. Rampon. On the calculation of transitive reduction-closure of orders. *Discreete Mathematics*, 111(1-3):289–303, February 1993.

[5] M. Hidalgo and R. Joan-Arinyo. Computing parameter ranges in constructive geometric constraint solving: Implementation and correctness proof. *Computer-Aided Design*, 44(7):709–720, 2012. DOI:10.1016/j.cad.2012.02.012.

[6] M. Hidalgo and R. Joan-Arinyo. The reachability problem in constructive geometric constraint solving based dynamic geometry. *Journal of Automated Reasoning*, 44(7):709–720, 2013. DOI:10.1007/s10817-013-9280-y.

[7] M. R. Hidalgo. *Geometric Constraint Solving in a Dynamic Geometry Framework*. PhD thesis, Universitat Politècnica de Catalunya, 2013.

[8] C.M. Hoffmann and R. Joan-Arinyo. A brief on constraint solving. *Computer-Aided Design and Applications*, 2(5):655–663, 2005.

[9] C.M. Hoffmann and K.J. Kim. Towards valid parametric CAD models. *Computer-Aided Design*, 33(1):376–408, 2001.

[10] C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Decompostion Plans for Geometric Constraint Problems, Part II: New Algorithms. *Journal of Symbolic Computation*, 31:409–427, 2001.

[11] C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Decompostion Plans for Geometric Constraint Systems, Part I: Performance Measurements for CAD. *Journal of Symbolic Computation*, 31:367–408, 2001.

[12] C. Jerman, G. Trombettoni, B. Neveu, and P. Mathis. Decomposition of geometric constraint systems: A survey. *International Journal of Computational Geometry and Applications*, 16(5-6):379–414, 2006.

[13] R. Joan-Arinyo and N. Mata. Applying constructive geometric constraint solvers to geometric problems with interval parameters. *Nonlinear Analysis*, 47(1):213–224, 2001.

[14] R. Joan-Arinyo and A. Soto. A correct rule-based geometric constraint solver. *Computer & Graphics*, 21(5):599–609, 1997.

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

[16] J.W. Klop. Term rewriting systems. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Background: Computational Structures*, volume 2 of *Handbook of Logic in Computer Science*, pages 1–117. Clarendon Press, 1992.

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

[18] S. Raghothama and V. Shapiro. Necessary conditions for boundary representations variance. In *Proceedings of the 13th International Annual Symposium on Computational Geometry*, pages 77–86, New York, 4-6 June 1997. ACM Press.

[19] S. Raghothama and V. Shapiro. Boundary representation deformation in parametric solid modeling. *ACM Transactions on Graphics*, 17(4):259–286, 1998.

[20] S. Raghothama and V. Shapiro. Consistent updates in dual representation systems. In *Proceedings of the Fith Symposium on Solid Modeling and Applications*, pages 65–75, New York, 9-11 June 1999. ACM Press.

[21] V. Shapiro and D.L. Vossler. What is a parametric family of solids? In *Proceedings of the Third Symposium on Solid Modeling and Applications*, pages 65–75, Salt Lake City, 17-19 May 1995. ACM Press.

[22] H.A. van der Meiden. *Semantics of Families of Objects*. PhD thesis, Delft Technical University, 2008.

[23] H.A. van der Meiden and W.F. Bronsvoort. A constructive approach to calculate parameter ranges for systems of geometric constraints. *Computer-Aided Design*, 38(4):275–283, 2006.