Transforming an Under-constrained Geometric Constraint Problem into a Well-constrained One

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

Universitat Politècnica de Catalunya
Escola Tècnica Superior d'Enginyeria Industrial de Barcelona
Av. Diagonal 647, 8a, E–08028 Barcelona
[robert, tonis, sebas, josep]@lsi.upc.es

ABSTRACT

We present an approach for handling geometric constraint problems with under-constrained configurations. The approach works by completing the given set of constraints with constraints that can be defined either automatically or drawn from an independently given set of constraints placed on the geometries of the problem. In both cases, the resulting completed set of constraints is not over-constrained. If every well-constrained subproblem in the given under-constrained configuration is solvable, the completed constraint problem is also solvable.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Geometric Algorithms, Languages, and Systems*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*Geometrical Problems and Computations*

General Terms

Theory, algorithms

Keywords

Constraint solving, geometric constraints, graph-based constraint solving, over-constrained and under-constrained systems

1. INTRODUCTION

Geometric constraint-based design is one of the main features of advanced computer-aided design systems and plays a paramount role as a basic tool to automate the product design cycle. In geometric constraint-based design the user inputs an sketch that approximately defines the shape of the object under design and, at some point, annotates the sketch with a set of constraints that precisely defines the intended object. It is left to the system to elucidate whether the object is well defined and, if so, the system generates some sort of representation that provides an effective way to build it. For a review on geometric constraint solving see [3] and references therein.

Existing geometric constraint solving techniques have been developed under the assumption that problems are well-constrained, that is, that the number of constraints and their placement on the geometric elements define a problem with a finite number of solutions for non-degenerate configurations.

There are a number of scenarios where the assumption of well-constrained does not apply. An example is the early stages of the design process when only a few parameters are established. The problem then is under-constrained, that is, it has an infinite number of solutions for non-degenerate configurations.

Another situation arises in cooperative design systems. Here, different activities in product design and manufacture examine different subsets of the information in the object's model. The presentation of such an information subset has been called a *view*, [2]. Maintaining views consistently is a central issue. To solve this problem, Hoffmann and Joan-Arinyo introduced in [7] the *constraint schema reconciliation* concept. It is assumed that with each view there is associated a geometric constraint graph (formally defined later). If the graphs are different, they are reconciled by drawing constraints from one of the graphs and adjoining them to the other graph. However, how to actually select the adjoined constraints is not defined.

To automate product design and manufacture, techniques to solve under-constrained problems and the schema reconciliation problem should be devised. If the geometric constraint problem is represented by a geometric constraint graph, solving under-constrained problems and the schema reconciliation problem can be seen as specific instances of the *geometric constraint graph completion* problem or just the *completion* problem. In terms of structurally well-constrained graphs, to be defined later, this problem is defined as follows:

Problem 1 (Well-constrained completion). *Given the geometric constraint graph associated to an struc-*

SM'03, June 16–20, 2003, Seattle, Washington, USA.

turally under-constrained geometric constraint problem, add automatically new edges (constraints) to the graph in such a way that the corresponding geometric constraint problem is structurally well-constrained.

Although not much attention has been paid to the well-constrained completion problem, its solution is not obvious at all. Results from distinct fields, such as combinatorial rigidity theory and matroid theory, must be taken into account to achieve an optimal solution for this problem. In Section 7 we briefly review this results and we discuss how to apply them to the solution of the well-constrained completion problem.

Only complete geometric constraint techniques solve any well-constrained geometric constraint graph, see [8]. The generality of complete solvers prevents them from always finding a symbolic construction plan. Then, numerical techniques must be applied to compute the solution. On the other hand, constructive geometric constraint solvers, like those in [4, 5, 17], are incomplete. However, they always compute a symbolic construction plan if the analyzed geometric constraint graph is in the domain of the solver. In this paper we study techniques to complete geometric constraint graphs. The result must be a graph that can be solved by constructive techniques. This problem, which is the object of this paper, is defined as follows:

PROBLEM 2 (COMPLETION). *Given the geometric constraint graph associated to an structurally under-constrained geometric constraint problem, add automatically new edges (constraints) to the graph in such way that the corresponding geometric constraint problem is solvable.*

We report here on a technique that solves Problem 2 in two different scenarios. In one, the extra constraints are automatically defined without any further condition. In the other one, the extra constraints are drawn from an independently given set of constraints defined on the geometries of the problem at hand. In the first scenario, the technique always generates a solvable problem if the initial problem is *completable*, i.e., all subproblems can be solved by a constructive technique.

In the second scenario, it is not always possible to generate a solvable graph, at least in a first step. This situation arises, for instance, when the set of extra constraints does not provide enough constraints. However, the algorithm always computes a completable graph, a graph that can be completed by applying the first technique.

The rest of the paper is organized as follows. Section 2 presents preliminary concepts on geometric constraint graphs, constructive geometric constraint solvers and its domain. The concept of tree decomposition of graphs is also discussed in order to characterize the domain of constructive techniques. Section 3 gives general definitions. Section 4 presents the general algorithm for completing a graph with extra constraints defined between the geometric elements with no further conditions. Section 5 presents the algorithms for completing a graph by adding constraints drawn from a given set. Empirical results for a suboptimal algorithm are given in Section 6. Well-constrained completion is discussed in Section 7. Section 8 presents three applications of the completion and a case study is developed. Finally we offer a brief summary in Section 9.

2. PRELIMINARIES

In this section we first discuss how geometric constraint problems can be represented by geometric constraint graphs and we recall necessary conditions for a graph to be well-constrained. Next, we focus on constructive geometric constraint solvers. This solvers analyze geometric constraint graphs and compute construction plans. Since it is not always possible to compute a construction plan for a well constrained graph we study the class of graphs solvable by constructive solvers. Finally, we recall the concept of tree decomposition of a graph. In addition to characterize the domain of constructive solvers, tree decompositions are useful to state the completion problem and its solutions.

2.1 Geometric Constraint Graphs

We consider geometric constraint problems in the Euclidean plane. The geometric elements are points, straight lines, and circles and arcs of circle with constant radii. The geometric constraints defined on these geometric elements include distance between two points, perpendicular distance between a point and a straight line, and angle between straight lines. Incidence, perpendicularity, parallelism, tangency and concentricity constraints can also be defined. These constraints can always be represented in terms of distance and angle constraints, [16].

In these conditions, a geometric constraint problem can be represented by a *geometric constraint graph*, $G = (V, E)$, where the vertices V are geometric elements with two degrees of freedom and the edges E are geometric constraints, each canceling one degree of freedom, [18]. Geometric constraint graphs are undirected simple graphs without loops or multiple edges. We assume that geometric constraint graphs have at least two vertices.

Figure 1 shows a mechanism defined as a geometric constraint problem. This mechanism, known as the crankshaft linkage, [1], transforms the linear motion of point p_5, along the straight line l_1, into a circular motion of point p_4 along a circle centered on point p_3. As a geometric constraint problem, the crankshaft linkage includes five points, p_i, $1 \leq i \leq 5$, and one straight line l_1. The set of geometric constraints are those listed in Figure 2 which includes point to point distance constraint, $dpp()$, and coincidence constraint, $on()$. The geometric constraint graph corresponding to the problem in Figure 1 is shown in Figure 3.

When a geometric constraint problem is represented by a geometric constraint graph we would like to know the graph properties that fulfill the geometric constraint graphs corresponding to well-constrained geometric constraint problems. A survey on the main results on combinatorial characterization of well-constrained graphs can be found in [18]. Unfortunately, no necessary and sufficient combinatorial conditions are known to decide whether or not a geometric constraint graph whose vertices represent points and lines, and whose edges represent distance and angle constraints is well-

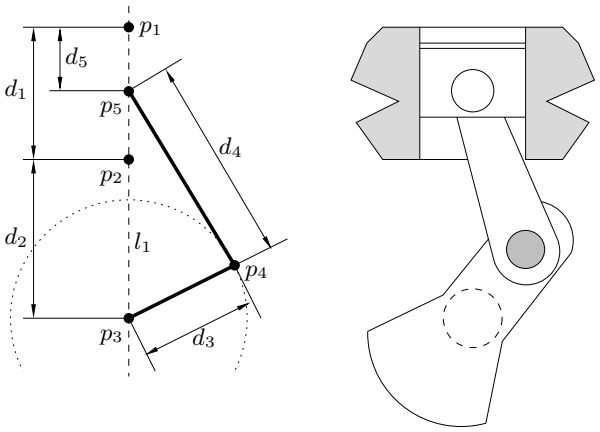

Figure 1: Crankshaft linkage and the actual mechanism.

constrained. Only necessary conditions are known for this kind of geometric constraint graphs. In this context, a technical definition useful for our purposes is the following, [5].

Definition 1. Let $G = (V, E)$ be a geometric constraint graph.

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

This definition gives necessary conditions for a graph to be well-constrained, [15, 21, 22, 23], therefore characterizes a superset of the class of well-constrained graphs.

Notice that the structurally over-constraint case handles the situation where the same graph is structurally over-constrained in one part and structurally under-constrained in another at the same time. To make the reading easier, from now on we will drop the word *structurally*.

2.2 Constructive Geometric Constraint Solvers

Many attempts to provide general, powerful and efficient methods for solving geometric constraint problems have been reported in the literature. For an extensive review in geometric constraint solving techniques refer to Fudos [5] and Durand [3].

1.	dpp(p_1, p_2, d_1)	6.	on(p_1, l_1)
2.	dpp(p_2, p_3, d_2)	7.	on(p_2, l_1)
3.	dpp(p_3, p_4, d_3)	8.	on(p_3, l_1)
4.	dpp(p_4, p_5, d_4)	9.	on(p_5, l_1)
5.	dpp(p_1, p_5, d_5)		

Figure 2: Geometric constraints for the crankshaft linkage.

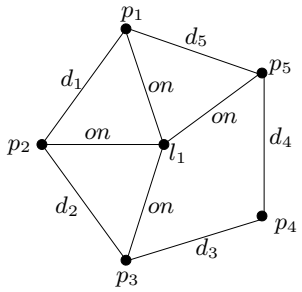

Figure 3: Geometric constraint graph for the crankshaft linkage.

Among the existing solving methods we focus on constructive solvers. In these solvers the input is a geometric constraint problem represented as a geometric constraint graph.

Constructive solvers describe the solution of a geometric constraint problem symbolically by means of a sequence of geometric constructions known as *construction plan*. Joan-Arinyo *et al.* give in [11] a detailed description of constructive solvers architecture. The most important component in constructive geometric constraint solvers is the *analyzer*. The analyzer computes a symbolic construction plan from the geometric constraint graph. Figure 4 shows the construction plan generated by a constructive solver for the crankshaft linkage in Figure 1. Geometric operations included in the plan are standard ruler-and-compass operations: intersection between lines and circles, construction of straight lines, construction of circles and so on.

In [8], Hoffmann *et al.* formalized the decomposition-recombination (DR)-planning problem as well as several performance measures by which DR-planning algorithms can be analyzed and compared. Among the performance measures described in [8] we are interested in completeness. A geometric constraint solver is complete if it solves all well-constrained problems.

Constructive geometric constraint solvers described in [4, 5, 17] are (DR)-planners according to Hoffmann *et al.* [8]. These solvers are incomplete, and, therefore, it makes sense to ask for their domain. We have characterized the domain of these methods in [10, 12, 13].

The domain characterization of these methods is based on the *tree decomposition* of a graph. In this manuscript the tree decomposition of a graph is used also to solve the geometric constraint graph completion problem. Therefore, the tree decomposition concept emerges as a tool interesting from both a theoretical and a practical point of view in constructive geometric constraint solving methods.

1.	$p_1 = pointXY(0, 0)$	6.	$c_2 = circleCR(p_1, d_5)$
2.	$p_2 = pointXY(0, d_1)$	7.	$p_5 = ilc(l_1, c_2)$
3.	$l_1 = line2P(p_1, p_2)$	8.	$c_3 = circleCR(p_3, d_3)$
4.	$c_1 = circleCR(p_2, d_2)$	9.	$c_4 = circleCR(p_5, d_4)$
5.	$p_3 = ilc(l_1, c_1)$	10.	$p_4 = icc(c_3, c_4)$

Figure 4: Construction plan for the crankshaft linkage.

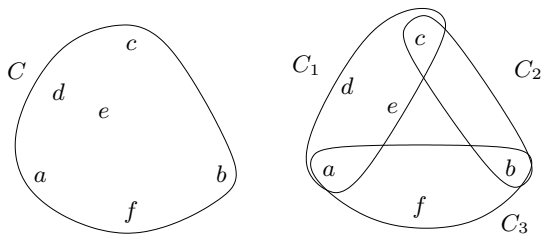

Figure 5: A set C (left) and a set decomposition of C (right).

2.3 Tree Decomposition

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

Definition 2. Let C be a set with, at least, three different members, say a, b, c. Let $\{C_1, C_2, C_3\}$ be three subsets of C. We say that $\{C_1, C_2, C_3\}$ is a *set decomposition* of C if

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

Figure 5 shows a set and a possible set decomposition. Next we define the concept of set decomposition of a graph.

Definition 3. Let $G = (V, E)$ be a geometric constraint graph, let $\{V_1, V_2, V_3\}$ be three subsets of V, and let E_1, E_2 and E_3 be the subsets of E corresponding to the subgraphs of G induced by V_1, V_2 and V_3 respectively. $\{V_1, V_2, V_3\}$ is a *set decomposition* of G if it is a set decomposition of V, and E_1, E_2 and E_3 are a partition of E.

Roughly speaking, a set decomposition of a graph $G = (V, E)$, is a set decomposition of the set of vertices V such that does not *break* any edge in E. Figure 6 illustrates this concept.

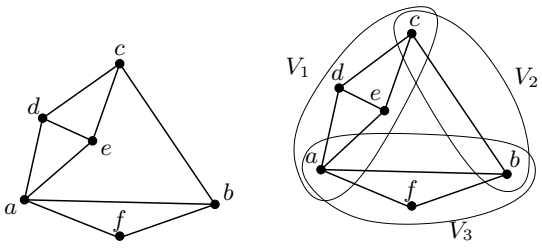

Figure 6: A graph G (left) and a set decomposition of G (right).

The concept of *tree decomposition* of a graph is defined as follows.

Definition 4. Let $G = (V, E)$ be a geometric constraint graph. A 3-ary tree T is a *tree decomposition* of G if

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

We will refer to a geometric constraint graph for which there is a tree decomposition as a *tree decomposable* graph. Tree decomposable graphs are not over-constrained. This is stated in the following lemma.

LEMMA 1. *Let $G = (V, E)$ be a geometric constraint graph. If G is tree decomposable, then G is not over-constrained.*

See [14] for the proof of lemmas and propositions in this section.

Figure 8 left shows an example of under-constrained graph and Figure 10 shows a tree decomposition of this graph. Notice that all the edges of the graph correspond to a leaf in the tree decomposition. The following proposition generalizes this observation.

PROPOSITION 1. *Let $G = (V, E)$ be a geometric constraint graph and T be a tree decomposition of G. For each edge $(a, b) \in E$, there is a leaf $\{a, b\}$ in T.*

Well-constrained graphs that are tree decomposable fulfill an additional property: there is a one-to-one correspondence between the edges in the graph and the leaves in the tree.

Definition 5. Let $G = (V, E)$ be a geometric constraint graph. T, a tree decomposition of G, is a *full tree decomposition* of G if there is a one-to-one correspondence between the leaves of T and the edges of G.

We will refer to a graph for which there is a full tree decomposition as a *full tree decomposable* graph.

LEMMA 2. *Let $G = (V, E)$ be a geometric constraint graph. G is well-constrained and tree decomposable if and only if G is full tree decomposable.*

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 which we will make use of later on.

PROPOSITION 2. *Let $G = (V, E)$ be a tree decomposable graph. For all $E' \subseteq E$, $G' = (V, E')$, the subgraph of G with the set of edges E', is tree decomposable.*

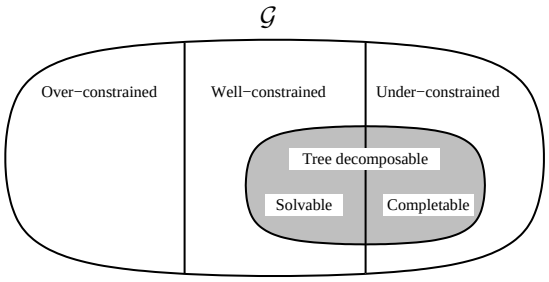

Figure 7: Classification of geometric constraint graphs according to Definition 1 and the set of tree decomposable graphs.

2.4 The domain of constructive techniques

In previous papers, [10, 12, 13], we proved that the domain of three constructive geometric constraint solving techniques is the same. Moreover, we proved that the domain of these techniques is the class of full tree decomposable graphs. We recall from [13] the following theorem. In this theorem, we say that a geometric constraint graph G is s-tree decomposable if it can be solved by Owen's algorithm, [17]; we say that G is solvable by reduction analysis if it can be solved by Fudos and Hoffmann's reduction analysis, [4]; and we say that G is solvable by decomposition analysis if it can be solved by Fudos and Hoffmann's decomposition analysis, [5], which was reformulated in [10].

THEOREM 1. *Let $G = (V, E)$ be a geometric constraint graph. The following assertions are equivalent:*

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

In what follows we will say that a constraint graph G is a *solvable* graph if it fulfills Theorem 1.

Theorem 1 states that the domain of constructive techniques is the class of solvable graphs. If $\mathcal{G}$ denotes the set of geometric constraint graphs, Figure 7 illustrates the relationship between different subsets of $\mathcal{G}$. The definitions of over-constrained, well-constrained and under-constrained graphs induce a partition on $\mathcal{G}$. By Lemma 1, tree decomposable graphs are a subset of those graphs which are not over-constrained. The class of tree decomposable graphs overlaps the set of well-constrained graphs and the set of under-constrained graphs. By Lemma 2, the solvable graphs (or full tree decomposable graphs) are the well-constrained graphs which are tree decomposable.

In this paper, we are concerned with the class of under-constrained graphs which are tree decomposable. We call the graphs in this class *completable* graphs. Sections 4 and 5 report on techniques to get solvable graphs from completable ones.

The proof of Theorem 1, [10, 12, 13], is constructive, in the sense that algorithms to compute tree decompositions can be derived from it. These algorithms compute a tree decomposition in polynomial time on the number of vertices of the graph. This time bound is justified because tree decompositions are computed in polynomial time from the output of the algorithms described in [4, 5, 17] which are polynomial.

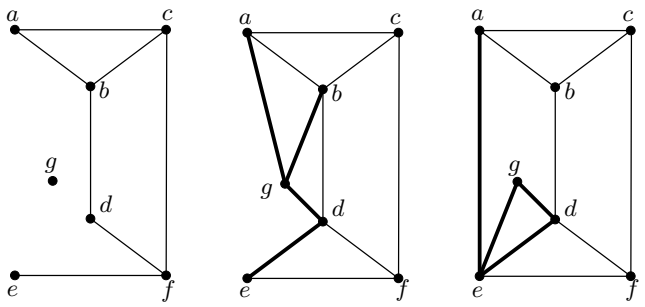

Figure 8: A graph and two different possible completions.

From now on, the solutions to the completion problem will be described in terms of tree decompositions. The main advantage of this strategy is that the proposed solutions do not depend on any particular algorithm used to solve the geometric constraint problem.

3. COMPLETABILITY OF UNDER-CONSTRAINED GRAPHS

Consider an under-constrained tree decomposable geometric constraint graph. To effectively solve it, first we should transform the graph into a solvable one. This amounts to adding new constraints to the graph, that is, adding new edges to the constraint graph.

If $G = (V, E)$ is the geometric constraint graph, Definition 1 gives the number of constraints that must be added to G to transform it into a well-constrained graph. However, deciding which constraints should actually be added to the graph is not an straightforward matter, because it could result in either an over-constrained graph or a well-constrained graph that is not solvable.

We will use the term *completion* to refer to the process of adding new constraints (edges) to an under-constrained graph. The resulting graph will be named the *completed* graph or just *a completion*.

Figure 8 left shows an under-constrained graph. Notice that there is just one edge incident to node e and that node g has no incident edges. Completion shown in the middle of Figure 8 is well-constrained and solvable. Completion in Figure 8 on the right side is well-constrained but not solvable.

We are interested in completed graphs that are solvable.

Definition 6. Let $G = (V, E)$ be an under-constrained geometric constraint graph. We say that G is *completable* if there is a solvable graph $G' = (V, E \cup E')$ which is a completion of G.

Algorithm FreeCompletion
1. Compute a tree decomposition T of G.
2. Compute the set of edges
$$E' = \{(a,b) \mid \{a,b\} \text{ is a leaf of } T \text{ and } (a,b) \notin E\}$$
3. Complete the graph G as $G' = (V, E \cup E')$.

Figure 9: Free completion algorithm for graph $G = (V,E)$.

In the following sections we present two techniques to complete completable graphs: free completion and conditional completion.

4. FREE COMPLETION

Here we present the first technique to complete completable graphs. We are given an under-constrained, completable graph $G = (V,E)$. The goal is to devise algorithms to automatically add new constraints to G to transform it into a solvable graph.

Among the different ways to complete the constraint graph G one can think of, we are interested first in one where the only additional requirement is that edges added to G must be taken from the set

$$E_F = \{(a,b) \mid a,b \in V \text{ and } (a,b) \notin E\}$$

We will call this a *free completion* of graph G.

The free completion problem has been previously addressed in [5]. There, a new method was developed to deal specifically with under-constrained problems. Our approach is based on tree decompositions and, since tree decompositions do not depend on any geometric constraint solving technique, is a general method.

The free completion algorithm is based on the following result.

PROPOSITION 3. *Let $G = (V,E)$ be an under-constrained geometric constraint graph. G is completable if and only if there is a tree decomposition of G.*

PROOF. Assume that $G = (V,E)$ is a completable graph. Then, there is a solvable graph $G' = (V, E \cup E')$. A solvable graph has a full tree decomposition. Let T be a full tree decomposition of G'. Since $E \subseteq E \cup E'$, by Proposition 2, G is tree decomposable.

Now assume that G is tree decomposable. Let T be a tree decomposition of G. Let E' be the set of leaves of T that do not correspond to an edge in E,

$$E' = \{(a,b) \mid \{a,b\} \text{ is a leaf of } T \text{ and } (a,b) \notin E\}.$$

The graph $G' = (V, E \cup E')$ is a completion of G and T is a full tree decomposition of G'. Thus, G is completable. □

The algorithm to carry out the free completion is given in Figure 9.

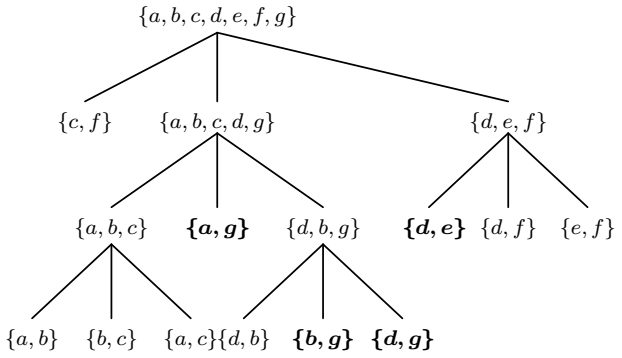

Figure 10: Tree decomposition of the graph in Figure 8 left.

Consider the constraint graph given in Figure 8 left and assume that step 1 in algorithm `FreeCompletion` computes the tree decomposition T shown in Figure 10. The leaves of T whose nodes do not define an edge in the graph depicted in Figure 8 left are (a,g), (d,e), (b,g) and (d,g). Therefore, the set of extra edges computed in step 2 is

$$E' = \{(a,g),(d,e),(b,g),(d,g)\}.$$

Figure 8 in the middle shows the constraint graph yielded by the algorithm `FreeCompletion`.

Notice that, in general, a tree decomposition T of a constraint graph $G = (V,E)$ is not unique. Therefore, a free completion of a graph G is not necessarily unique.

5. CONDITIONAL COMPLETION

Conditional completion is the second technique to complete geometric constraint graphs presented in this paper. The study of conditional completion was originally motivated by the constraint schema reconciliation problem, [7]. Section 8.1 briefly discusses this problem and how conditional completion allows to solve it.

Let $G = (V,E)$ be a completable geometric constraint graph. Let $\widehat{G} = (V, \widehat{E})$ be a geometric constraint graph whose edges $\widehat{E}$ define a set of additional constraints between geometric elements in V with $\widehat{E} \cap E = \emptyset$. Notice that if $\widehat{E} \cap E \neq \emptyset$, we always can consider as additional edges those in the set $\widehat{E}' = \widehat{E} \setminus E$.

Our interest is to build a solvable completion of G by adding edges drawn from $\widehat{E}$. We will refer to this completion as *conditional completion* of graph G from $\widehat{G}$. Formally, we define a conditional completion as follows.

Definition 7. Let $G = (V,E)$ be a completable geometric constraint graph and let $\widehat{G} = (V, \widehat{E})$ be a geometric constraint graph. $G' = (V, E')$ is a *conditional completion* of G from $\widehat{G}$ if G' is a completable completion of G and $E' = E \cup \widehat{E}'$ with $\widehat{E}' \neq \emptyset$ and $\widehat{E}' \subseteq \widehat{E}$.

We call the graph $\widehat{G}$ the *additional graph*. Figure 11 shows an under-constrained graph G (left) and an additional graph $\widehat{G}$ (right). Figure 12 shows two conditional completions of

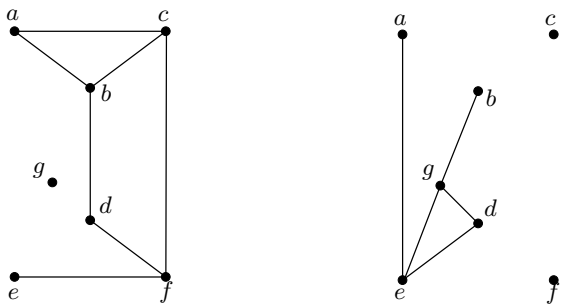

Figure 11: Geometric constraint graphs G (left) and $\widehat{G}$ (right).

G from $\widehat{G}$. The graph on the left is completable and the graph on the right is solvable.

Among all the possible conditional completions of an under-constrained graph with respect to a given additional graph, we are interested in those that are maximal in the following sense.

Definition 8. Let $G' = (V, E')$ be a conditional completion of G from $\widehat{G}$. We say that G' is a *maximal conditional completion* if for any other conditional completion $G'' = (V, E'')$ from $\widehat{G}$, the relation $|E''| \leq |E'|$ holds.

A maximal conditional completion does not need to result in a solvable graph. Notice that it is not possible to build a solvable completion if the number of edges in $\widehat{E}$ is smaller than the number of edges required by Definition1 to transform G into a well-constrained graph. In these conditions, however, a solvable graph still can be built applying free completion to the graph yielded by the maximal conditional completion.

5.1 Maximal Conditional Completion as a Combinatorial Optimization Problem

Following Papadimitriou *et al.* [20], a *subset system* $S = (\mathcal{E}, \mathcal{S})$ is a finite set $\mathcal{E}$ together with a collection $\mathcal{S}$ of

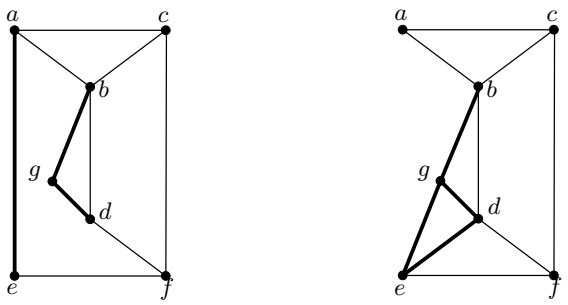

Figure 12: Conditional completion (left) and maximal conditional completion (right) of G from $\widehat{G}$ in Figure 11.

Algorithm GreedyOptimize
1. $I := \emptyset$
2. while $\mathcal{E} \neq \emptyset$ do
2.1 $e :=$ a maximum weight member of $\mathcal{E}$
2.2 $\mathcal{E} := \mathcal{E} \setminus \{e\}$
2.3 if $I \cup \{e\} \in \mathcal{S}$ then $I := I \cup \{e\}$

Figure 13: The greedy algorithm for subset systems.

subsets of $\mathcal{E}$ closed under inclusion (that is, if $A \in \mathcal{S}$ and $A' \subseteq A$, then $A' \in \mathcal{S}$). The elements of $\mathcal{S}$ are called *independent*. The *combinatorial optimization problem associated with* a subset system $(\mathcal{E}, \mathcal{S})$ is the following: Given a *weight* $w(e) \geq 0$ for each $e \in \mathcal{E}$, find an independent subset that has the largest possible total weight.

Let $G = (V, E)$ be an under-constrained graph and let $\widehat{G} = (V, \widehat{E})$ be the additional graph. Assume that E and $\widehat{E}$ are disjoint, $E \cap \widehat{E} = \emptyset$. If we define $\mathcal{E} = E \cup \widehat{E}$ and $\mathcal{S}$ as the subsets of $\mathcal{E}$ such that are tree decomposable, then S is a subset system. We say that a set of edges $E' \subseteq \mathcal{E}$ is tree decomposable if the graph $G' = (V, E')$ is tree decomposable. Notice that by Proposition 2, $\mathcal{S}$ is closed under inclusion.

We define the weight function $w(e)$ over the set of edges $\mathcal{E} = E \cup \widehat{E}$ as

$$w(e) = \begin{cases} 1 & \text{if } e \in \widehat{E} \\ 2 & \text{if } e \in E \end{cases}$$

In these conditions, the problem of computing a maximal conditional completion of G from $\widehat{G}$ is a combinatorial optimization problem associated with the subset system S and the weight function w.

5.2 The Greedy Algorithm

The greedy algorithm is the simplest algorithm to solve a combinatorial optimization problem associated with a subset system. In this section we first describe the greedy algorithm for subset systems. Then we apply it to the maximal conditional completion problem.

Figure 13 shows the greedy algorithm on the subset system $S = (\mathcal{E}, \mathcal{S})$. The algorithm computes I, an independent set in S which we want to have the largest total weight. We recall from [20] that it is not necessary to compute explicitly S to answer the question $I \cup \{e\} \in \mathcal{S}$ in the step 2.3 of the algorithm. It suffices to have an algorithm to decide whether the set $I \cup \{e\}$ is in S.

Since we have formalized the maximal conditional completion of a graph G from an additional graph $\widehat{G}$ as a combinatorial optimization problem, we can use the greedy algorithm in Figure 13 as is, without any change. Notice that, for the maximal conditional completion problem, the elements of $\mathcal{S}$ are the subsets of $\mathcal{E}$ such that they are tree decomposable. Therefore, deciding whether $I \cup \{e\}$ is in $\mathcal{S}$ is equivalent to decide whether there is a tree decomposition of the set of edges $I \cup \{e\}$.

Unfortunately, the greedy algorithm does not solve the maximal conditional completion problem. Let us show a coun-

terexample. Let G be the completable graph in Figure 11 left and let $\widehat{G}$ be the geometric constraint graph in Figure 11 right. We define the subset system S and the weight function w as in Section 5.1.

The greedy algorithm in Figure 13 first selects the edges in G because their weights are larger than the weights of edges in $\widehat{G}$. Hence

$$I = \{(a,b),(a,c),(b,c),(b,d),(c,f),(d,f),(e,f)\}$$

After that, the edges in $\widehat{G}$ are considered. Because all the edges in $\widehat{G}$ have the same weight, they can be chosen at random. Assume that the sequence of edges chosen is (a,e), (b,g) and (d,g). Then, no more edges can be added to I and the independent set returned by the greedy algorithm is

$$\begin{aligned} I &= \{(a,b),(a,c),(b,c),(b,d),(c,f),(d,f),(e,f),\\ &\quad (a,e),(b,g),(d,g)\} \end{aligned}$$

See Figure 12 left.

Now assume that the sequence of chosen edges is (e,d), (e,g), (d,g) and (g,b). At this point, no more edges can be added to I and the independent set returned by the greedy algorithm is

$$\begin{aligned} I &= \{(a,b),(a,c),(b,c),(b,d),(c,f),(d,f),(e,f),\\ &\quad (e,d),(e,g),(d,g),(g,b)\} \end{aligned}$$

See Figure 12 right. Clearly, this independent set includes more edges than the one computed before, that is, it has larger total weight. Therefore, the greedy algorithm does not necessarily yield a maximal conditional completion.

6. EXPERIMENTAL STUDY OF THE GREEDY ALGORITHM FOR THE CONDITIONAL COMPLETION PROBLEM

The greedy algorithm does not always solve the maximal conditional completion problem. Therefore, it makes sense to investigate how far the completions yielded by the greedy algorithm are from being maximal.

Let $G = (V,E)$ be an under-constrained graph and let $\widehat{G} = (V,\widehat{E})$ be the additional graph from where G must be completed. Let $\mathcal{G}(G,\widehat{G})$ denote the number of extra edges actually added to G from $\widehat{G}$ by the greedy algorithm in an specific run. Recall that since extra edges with the same weight are selected at random, the greedy algorithm is not deterministic. Let $\mathcal{O}(G,\widehat{G})$ denote the number of extra edges taken from $\widehat{G}$ that results in a maximal completion of G.

The goal of the experimental study is to measure how different $\mathcal{G}(G,\widehat{G})$ and $\mathcal{O}(G,\widehat{G})$ are for a large enough set of pairs $(G,\widehat{G})$. But computing $\mathcal{O}(G,\widehat{G})$ entails having an algorithm that solves the maximal completion problem. Therefore, we compute an estimation for $\mathcal{O}(G,\widehat{G})$.

Two different tests were conducted depending on the strategy used to define the additional set of edges.

6.1 First test

6.1.1 Test Definition

The pairs of graphs $G(V,E)$ and $\widehat{G} = (V,\widehat{E})$, with $|V| = n$, have been defined as follows:

1. A constructively solvable graph $F = (V,E_F)$ with $|V| = n$ is randomly generated. This is done by building a random tree decomposition T of $G(V,E)$ with $E = \emptyset$. Then E_F is defined as
$$E_F = \{(a,b) | a,b \in V \text{ and } (a,b) \text{ is a leaf of } T\}$$
2. The set E_F is randomly split into three pairwise disjoint subsets E, D, and O such that $E_F = E \cup D \cup O$.
3. Let K_n be the complete graph with n vertices. Let A be a set of edges randomly drawn from $E(K_n) \setminus E$. Then $\widehat{E}$ is defined as $\widehat{E} = O \cup A$.

Notice that by construction of G and $\widehat{G}$, all the edges in $O \subseteq \widehat{E}$ can be added to G. Therefore the following inequality holds:

$$0 \leq |O| \leq \mathcal{O}(G,\widehat{G}) \leq 2|V| - 3 - |E|$$

Thus, if $\mathcal{G}(G,\widehat{G}) < |O|$ then $\mathcal{G}(G,\widehat{G}) < \mathcal{O}(G,\widehat{G})$. This means that the completion yielded by the greedy algorithm is not maximal.

6.1.2 Data Description

A series of sets of graphs each containing twenty pairs $(G = (V,E), \widehat{G} = (V,\widehat{E}))$ where generated. Values of $|V| = n$ were $\{5,6,\ldots,100\}$. The greedy algorithm was serially fed with each pair. For each run, the values of $|E|$, $|\widehat{E}|$, $|O|$, and the number of edges actually added to G, $\mathcal{G}(G,\widehat{G})$ were recorded.

6.1.3 Results Analysis

The ratio of runs for which the completion was not maximal, that is, the number of edges actually added to G, $\mathcal{G}(G,\widehat{G})$, was smaller than $|O|$, with respect the total number of runs was 3.4%.

The difference between $\mathcal{G}(G,\widehat{G})$, and the number of edges that could be added, $|O|$, was measured by the ratio of missing edges

$$\frac{|O| - \mathcal{G}(G,\widehat{G})}{|O|}$$

Figure 14 shows the number of runs versus the ratio of missing edges. For example, there were 13 runs where the ratio was 2%, 10 runs with a ratio of 3% and so on.

Finally the ratio of cases where the resulting completion is well-constrained, $|E \cup \widehat{E}| = 2|V| - 3$, with respect the total number of runs was 7.8%.

6.2 Second Test

6.2.1 Test Definition

The pairs of graphs $(G(V,E), \widehat{G}(V,\widehat{E}))$ are defined as follows:

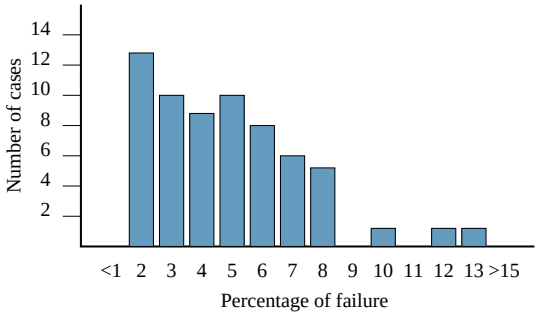

Figure 14: Distribution of runs where more edges in $\widehat{G}$ could be added to G.

1. A constructively solvable graph $F = (V, E_F)$ is built as described for the first test.
2. The set of edges E_F is split into two subsets E and D with $E \cap D = \emptyset$ and $E_F = E \cup D$.
3. If K_n is the complete graph with n vertices, the additional set of edges is now $\widehat{E} = E(K_n) \setminus E$.

Notice that since all the edges needed to transform the under-constrained graph G into a well-constrained one are in $\widehat{E}$, the number of edges that can be added is given by $\mathcal{O}(G, \widehat{G}) = 2|V| - 3 - |E|$. Therefore, for this test the following inequality holds:

$$0 \leq \mathcal{G}(G, \widehat{G}) \leq \mathcal{O}(G, \widehat{G}) = 2|V| - 3 - |E|$$

6.2.2 Data Description

A series of sets of graphs each containing ten pairs $(G = (V, E), \widehat{G} = (V, \widehat{E}))$ where generated. Values of $|V| = n$ were $\{5, 10, 15, \ldots, 100\}$. The data recorded for each run was the same as in the first test, $|E|$, $|\widehat{E}|$, $|O|$, and $\mathcal{G}(G, \widehat{G})$.

6.2.3 Results Analysis

In this test we counted the number of runs $(G, \widehat{G})$ for which $\mathcal{G}(G, \widehat{G}) < \mathcal{O}(G, \widehat{G})$. That is the number of runs where the greedy algorithm did not yield a well-constrained completion. There were exactly 3 cases, that is 1.5% of the runs.

6.3 Experimental Study Conclusions

From the experimental study we can conclude that:

1. The greedy algorithm does not compute the maximal conditional completion in a small fraction of cases. This fraction can be estimated in about 3.5%.
2. When the greedy algorithm does not find a maximal conditional completion, the ratio of missing edges needed to generate a maximal completion with respect to the total number of edges in the well-constrained graph was less than 8%.
3. If the additional set of edges includes as many edges as needed to generate a well-constrained completion, only in 1.5% of the runs the completion yielded was under-constrained.

Therefore, the greedy algorithm is of practical application to compute conditional completions.

7. WELL-CONSTRAINED CONDITIONAL COMPLETION

So far, we have focused our interest in completing a geometric constraint graph in such a way that the completed graph was solvable by a constructive technique. Here we relax this requirement and we only expect the completion to be well-constrained; that is, we address Problem 1.

We proceed as in Section 5. First, we define the well-constrained conditional completion. The main difference with respect to Definition 7 is that here the geometric constraint graph is required to be under-constrained instead of completable.

Definition 9. Let $G = (V, E)$ be an under-constrained geometric constraint graph and let $\widehat{G} = (V, \widehat{E})$ be a geometric constraint graph. $G' = (V, E')$ is a *well-constrained conditional completion* of G from $\widehat{G}$ if G' is a well-constrained completion of G and $E' = E \cup \widehat{E}'$ with $\widehat{E}' \neq \emptyset$ and $\widehat{E}' \subseteq \widehat{E}$.

Next, we define the maximal well-constrained conditional completion. Notice the analogy with Definition 8.

Definition 10. Let $G' = (V, E')$ be a well-constrained conditional completion of G from $\widehat{G}$. We say that G' is a *maximal well-constrained conditional completion* if for any other well-constrained conditional completion $G'' = (V, E'')$ from $\widehat{G}$, the relation $|E''| \leq |E'|$ holds.

Now, we reformulate the maximal well-constrained conditional completion problem as a combinatorial optimization problem associated with a subset system. Let $G = (V, E)$ be an under-constrained graph and let $\widehat{G} = (V, \widehat{E})$ be the additional graph. Assume that E and $\widehat{E}$ are disjoint, $E \cap \widehat{E} = \emptyset$. If we define $\mathcal{E} = E \cup \widehat{E}$ and $\mathcal{S}$ as the subsets E' of $\mathcal{E}$ for which the graph $G' = (V(E'), E')$ is not over-constrained, then S is a subset system. We define the weight function $w(e)$ over the set of edges $\mathcal{E}$ as in Section 5.1. $V(E')$ denotes the set of the endpoints of the edges in E'.

Two well established results are needed to show that now the greedy algorithm solves the maximal well-constrained conditional completion problem. These results come from matroid theory and combinatorial rigidity theory, respectively.

Matroids are subset systems which fulfill some additional properties. See [19] for an in depth study of matroid theory. A well known result in combinatorial optimization is that the greedy algorithm solves any combinatorial optimization problem associated with a subset system which is a matroid, [20].

Geometric constraint graphs whose vertices are points in the plane and whose edges are distance constraints has been extensively studied in the context of combinatorial rigidity theory. The result we are interested in is the following, [6].

THEOREM 2. *Let $G = (V, E)$ be a geometric constraint graph whose vertices are points in the plane and whose edges are distance constraints. Let $\mathcal{E} = E$ and let $\mathcal{S}$ denote the collection of subsets of $\mathcal{E}$ such that they are not over-constrained. Then the subset system $S = (\mathcal{E}, \mathcal{S})$ is a matroid.*

Combining these results, we can conclude that the greedy algorithm solves the maximal well-constrained conditional completion for graphs whose vertices are points in the plane and whose edges are distance constraints. This result has interesting practical implications. For example, any complete geometric constraint solving technique can benefit from the use of the greedy algorithm to solve the maximal well-constrained conditional completion problem. Moreover, it can also solve over-constrained problems by applying the technique described in Section 8.3.

8. APPLICATIONS OF THE CONDITIONAL COMPLETION

We present three applications of the maximal conditional completion problem: constraint schema reconciliation, constraints with priorities and solving over-constrained problems. We also include a case study consisting in an over-constrained problem in which we require the topological constraints to be kept in the final solution.

8.1 Constraint Schema Reconciliation

In cooperative design systems, different activities in product design and manufacture aim at different subsets of the information in the model under design. Each of those subsets of information has been called a *view*, [2].

In this context, maintaining views consistently is a central issue. To solve this problem, Hoffmann and Joan-Arinyo introduced in [7] the *constraint schema reconciliation* concept. There, the authors assume that each view has an associated geometric constraint graph. If the graphs are different, they are reconciled by drawing constraints from one of the graphs and adjoining them to the other graph. However, how to actually select the constraints to be drawn is not explicitly addressed in [7].

Joan-Arinyo *et al.* reported in [9] on a framework to support geometric constraint-based design systems with multiple views for concurrent engineering. The framework is based on a conceptual architecture with a master view and several client views with a two-way flow information between the master and client views. The framework addresses the problem of maintaining consistency between different views and specifically the constraint schema reconciliation.

Figure 15 shows an example of constraint schema reconciliation in the framework described in [9]. Let M be a master model. First, a new client view V is opened. The new view V does not include dimensional constraints but has the same geometry and topology as M. Then, the designer adds new constraints to the view V according to its specific needs. Notice that the resulting view V' is usually under-constrained. Next, the view V'' is obtained by computing a maximal conditional completion of the view V' from the master model

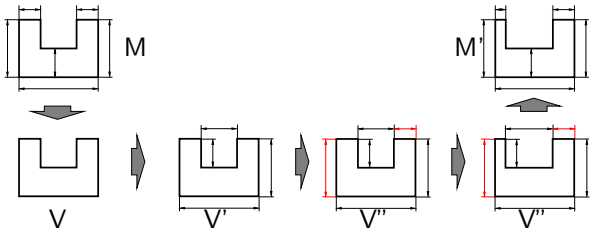

Figure 15: Constraint schema reconciliation.

M and, possibly, applying free completion. Finally, the designer changes the values of dimensional constraints in the view V''. When the editing is over, the master model M is updated from view V''.

8.2 Constraints with priorities

In constraint-based design there are situations where it would be useful to associate a different level of priority to different constraints. For example when there is a set of mandatory constraints but the designer still has the freedom to fix the remaining constraints. We can generalize the maximal conditional completion problem and include constraints with levels of priority.

Let M be the maximum level of priority. Let E_i, $1 \leq i \leq M$, be a partition of the set of edges $\mathcal{E}$. Then, we have a collection of graphs $G_i = (V, E_i)$, $1 \leq i \leq M$, on the same set of vertices V. V contains the endpoints of the edges in $\mathcal{E}$. Let $\mathcal{S}$ be the set of subsets of $\mathcal{E}$ which are tree decomposable. Then $S = (\mathcal{E}, \mathcal{S})$ is a subset system. If for each $e \in \mathcal{E}$, we define the weight function $w(e) = i$ if $e \in E_i$, we get a combinatorial optimization problem on the subset system S.

Then the greedy algorithm in Figure 13 applies. If $G_M = (V, E_M)$ is completable, the constraints in E_M are guaranteed to be in the resulting graph. The edges on the remaining sets E_j, for $1 \leq j < M$, are successively included in the resulting graph according to the j-th level of priority.

8.3 Over-constrained problems

We first state what we understand by solving an over-constrained problem.

Definition 11. Let $G^* = (V, E^*)$ be an over-constrained geometric constraint graph associated to an over-constrained problem. The geometric constraint graph $G = (V, E)$ is a *solution of the over-constrained problem* if E is a subset of E^* with the largest possible cardinality such that G is completable.

Let $G^* = (V, E^*)$ be an over-constrained graph. We define the graphs G and $\widehat{G}$ as follows.

$$\begin{aligned} G &= (V, \emptyset) \\ \widehat{G} &= (V, E^*) \end{aligned}$$

Now, we apply the greedy algorithm to compute a maximal conditional completion of G from $\widehat{G}$. If the resulting graph is completable, a free completion can be applied to get a

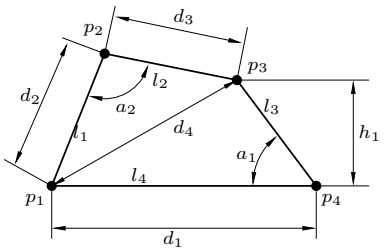

Figure 16: Over-constrained sketch

solvable graph. Recall that an over-constrained graph has a subgraph with more edges than needed, but it can have subgraphs with less edges than needed. Therefore, we can not get, in general, a solvable graph just by selecting a subset of edges in E^*.

Notice that due to the non-deterministic nature of the greedy algorithm each run could result in a different completable (or solvable) graph.

8.4 Case study

To illustrate how the ideas presented here work, consider the over-constrained problem shown in Figure 16. The geometric elements in the problem include four points and four straight segments. The constraints are four point-point distances, d_1, d_2, d_3, d_4, two angles, a_1, a_2, and one point-segment distance, h_1.

Assume that the over-constrained graph $G^* = (V, E^*)$ associated with the sketch is the one given in Figure 17. And assume that, with the aim of keeping the sketch topology, the following priorities are defined on the constraints

$$w(e) = \begin{cases} 1 & \text{if } e \in E^* \text{ is a dimensional constraint} \\ 2 & \text{if } e \in E^* \text{ is a topological constraint} \end{cases}$$

If graphs G and $\widehat{G}$ are defined as in Section 8.3, Figure 18 shows a constraint graph output by the greedy algorithm. Recall that the greedy algorithm is not deterministic and that the actual output depends on the specific run.

Now, a different set of constraints, which is constructively solvable, can be suggested to the end user to annotate the input sketch. This set of constraints is shown in Figure 19.

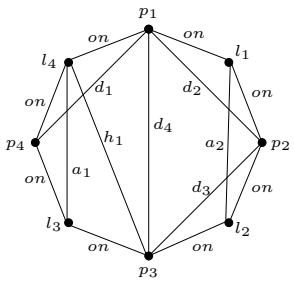

Figure 17: Over-constrained graph associated with the problem in Figure16.

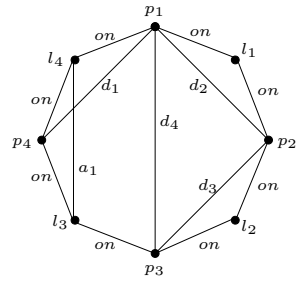

Figure 18: Geometric constraint graph built by the greedy algorithm.

9. SUMMARY

The completion of a geometric constraint graph consists in adding extra edges to an under-constrained graph. In this paper we have addressed two techniques to solve the completion problem: free completion and maximal conditional completion. In free completion, the extra constraints are automatically defined without any further condition. In maximal conditional completion, the extra edges are drawn from the set of edges of a second graph which we call the additional graph. The resulting graph in both cases must also fulfill an additional property: it must be solvable by a constructive geometric constraint solving technique.

We have reformulated the maximal conditional completion problem as a combinatorial optimization problem associated with a subset system. In this context, we have investigated the applicability of the greedy algorithm. Although we have shown that the greedy algorithm does not guarantee the solution of the maximal conditional completion problem, the experimental results reveal that the greedy algorithm can be successfully used to compute conditional completions.

We have also revised the maximal well-constrained completion problem. Here, we only require the resulting graphs to be well-constrained instead of solvable. Recalling standard results from matroid theory and combinatorial rigidity theory, we have shown that the greedy algorithm solves this problem.

Finally, we have presented three applications of the maximal conditional completion problem. First, we have shown how the maximal conditional completion problem can be used in the solution of the schema reconciliation problem in cooperative design. This problem motivated the study

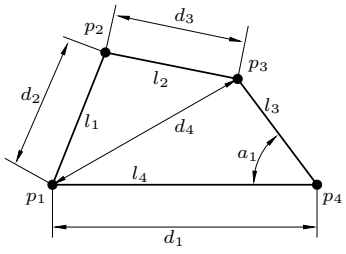

Figure 19: Solvable problem for the the graph in Figure 18.

of the maximal conditional completion. Second, we have presented a generalization of the maximal conditional completion problem in which the set of additional constraints is structured in priority levels. And third, we have discussed the applicability of the maximal conditional completion to deal with over-constrained geometric constraint problems.

Acknowledgements

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

10. REFERENCES

[1] H. Cundy and A. Rollett. *Mathematical Models*. Oxford University Press, 2nd edition, 1961.

[2] K. de Kraker, M. Dohmen, and W. Bronsvoort. Multiple-way feature conversion to support concurrent engineering. In M. Pratt, R. Siriram, and M. Wozny, editors, *Product Modeling for Computer Integrated Design and Manufacture*, pages 203–212. Chapman and Hall, London, UK, 1997.

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

[4] I. Fudos and C. Hoffmann. Correctness proof of a geometric constraint solver. *International Journal of Computational Geometry & Applications*, 6(4):405–420, 1996.

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

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

[7] C. Hoffmann and R. Joan-Arinyo. Distributed maintenance of multiple product views. *Computer-Aided Design*, 32(7):421–431, June 2000.

[8] C. 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.

[9] R. Joan-Arinyo, A. Soto, S. Vila, and J. Vilaplana. A framework to support multiple views in geometric constraint-based models. In E. Dekneuvel, editor, *Proceedings of the 8th. IEEE International Conference on Emerging Technologies and Factory Automation ETFA'2001*, Antibes-Juan les Pins, France, Oct. 2001. 8th. IEEE.

[10] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana. On the domain of constructive geometric constraint solving techniques. In R. Ďurikovič and S. Czanner, editors, *Spring Conference on Computer Graphics*, pages 49–54. IEEE Computer Society, 2001.

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

[12] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana. Revisiting decomposition analysis of geometric constraint graphs. In K. Lee and M. Patrikalakis, editors, *Seventh ACM Symposium on Solid Modeling and Applications*, pages 105–115, Saarbrüken, Germany, 2002. ACM Press.

[13] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana. Revisiting decomposition analysis of geometric constraint graphs. *Computer-Aided Design*, 2003. to appear.

[14] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana-Pastó. Transforming an under-constrained geometric constraint problem into a well-constrained one. Research LSI-02-69-R, Dept. Llenguatges i Sistemes Informàtics, UPC, Barcelona, Nov. 2002.

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

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

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

[18] J. Owen. Constraints on simple geometry in two and three dimensions. *International Journal of Computational Geometry & Applications*, 6(4):421–434, 1996.

[19] J. Oxley. *Matroid Theory*. Oxford University Press, 1992.

[20] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms And Complexity*. Dover Publications, May 1998.

[21] B. Servatius and W. Whiteley. Constraining plane configurations in computer-aided design: combinatorics of directions and lengths. *SIAM Journal on Discrete Mathematics*, 12(1):136–153 (electronic), 1999.

[22] K. Sugihara. On some problems in the design of plane skeletal structures. *SIAM Journal on Algebraic and Discrete Methods*, 4(3):355–362, 1983.

[23] W. Whiteley. Some matroids from discrete applied geometry. In J. Bonin, J. Oxley, and B. Servatius, editors, *Matroid Theory*, volume 197 of *Contemporary Mathematics*. American Mathematical Society, 1996.