International Journal of Computational Geometry & Applications
© World Scientific Publishing Company

Well-constrained Completion and Decomposition for Under-constrained Geometric Constraint Problems

Gui-Fang Zhang
School of Sciences, Beijing Forestry University, Beijing 100083, China

Xiao-Shan Gao
KLMM, Institute of Systems Science, AMSS, Academia Sinica, Beijing 100080, China

Received (received date)
Revised (revised date)
Communicated by (Name)

In this paper, we consider the optimal well-constrained completion problem, that is, for an under-constrained geometric constraint problem, add automatically new constraints in such a way that the new constraint problem G is well-constrained and the set of equations to be solved simultaneously in order to solve G has the smallest size. We propose a polynomial time algorithm which gives a partial solution to the above problem.

Keywords: Geometric constraint solving, under-constrained completion, decomposition.

1. Introduction

Geometric constraint solving (abbr. GCS) is one of the key techniques in intelligent and parametric CAD, which allows the user to make modifications to existing designs by changing parametric values. GCS methods may also be used in other fields like computer vision, molecular modelling, robotics, and feature-based design. There are four major approaches to GCS: the numerical approach, the symbolic approach, the rule-based approach, and the graph-based approach.

Most existing GCS methods assume that the problems are well-constrained. This paper will focus on using graph-based algorithms to solve under-constrained problems. The reason behind this research is that a natural way to draw a design figure is to do it incrementally, that is, to add geometric primitives and geometric constraints one by one. After a new primitive or a new constraint is added, it would be better for the constraint solving system to generate the design diagrams. But, before all the necessary constraints are added, the constraint problem is under-constrained.

Joan-Arinyo et al suggested that the main problems for solving under-constrained problems are *well-constrained completion* and *completion.*[1]

Problem 1.1. [**Well-constrained Completion**] Given the geometric constraint graph associated to an under-constrained geometric constraint problem, add automatically new constraints to the graph in such a way that the corresponding geometric constraint problem is well-constrained.

Problem 1.2. [**Completion**] Given the geometric constraint graph associated to an under-constrained geometric constraint problem, add automatically new constraints to the graph in such a way that the corresponding geometric constraint problem is solvable, that is, the corresponding geometric constraint problem can be solved by geometric constructions, like ruler and compass constructions.

Since not all under-constrained problems have solvable completions, we propose the following more general problem.

Problem 1.3. [**Optimal Well-constrained Completion**] Given the geometric constraint graph associated to an under-constrained geometric constraint problem, add automatically new constraints to the graph in such a way that the corresponding geometric constraint problem G is well-constrained and the set of equations to be solved simultaneously in order to solve G has the smallest size.

Latham and Middleditch gave a method to detect whether the constraint graph in 2D or 3D is over- or under-constrained and decide how to delete or add constraints to make it well-constrained.[2] Therefore, Latham and Middleditch solved Problem 1.1. But the part on how to add new constraints is not studied in detail. So their method does not address Problems 1.2 and 1.3. Fudos and Hoffmann proposed a method which could handle those problems that can be solved with the so-called cluster formation method.[3] Yuan et al proposed an algorithm based on local propagation. Their algorithm can be used to find the available solutions according to the design intention in certain sense, but cannot deal with constraint problems with so-called cyclic constraints.[4] Lee et al gave a method to deal with under-constrained problems in 2D by classifying the under-constrained subgraphs into simplified cases and by applying some classification rules.[5] Joan-Arinyo et al proposed algorithms to solve the well-constrained completion problem and the completion problem in 2D with the technique of s-tree decomposition.[1,6] This method can handle those problems that can be solved with the s-tree decomposition method. In this volume, Trombettoni et al used a degree-freedom-analysis to solve under-constrained problems.[7] Their method is polynomial in time but seems cannot generate optimal completions for problems like the one in Figure 10.

In this paper, we will give an improved solution to Problem 1.1 and a partial solution to Problem 1.3. The main ideas of the algorithms will be explained below.

First, we observe that fixing the position of several geometric primitives called *base primitives* could lead to better well-constrained completions. So our algorithm will try to solve the problem starting with a set of base primitives and if we cannot find a decomposition with these base primitives, we will repeat the process with a new set of base primitives. Since the number of base primitives is less than the number of constraints, this idea leads to a systematical and effective search of optimal well-constrained completions. After a set of base primitives is fixed, we will generate a general construction sequence (GC) (definition in Section 2) for the problem, which gives a solving order for the primitives. So, a natural way to add a new constraint from o_1 to o_2 is that o_1 is before o_2 in the sequence. In this way, we will not destroy the solving order. Another idea is that whenever possible, we will construct the primitives one by one. We show that in the 2D case, all the newly added constraints can be used for this purpose. Based on the above ideas, we give an improved algorithm for Problem 1.1 in Section 3. The complexity of the algorithm is $O(n^2)$ where n is the number of the primitives in the problem.

The well-constrained completion algorithm in Section 3 is not optimal, that is, it does not solve Problem 1.3. In order to improve the output of the algorithm, a D-tree decomposition algorithm for under-constrained problem is proposed. During the well-constrained completion, we will check whether the constraint problem can be split into two smaller problems, and if not, we will select a set of new base primitives and repeat the well-constrained completion again. This part is presented in Section 4 and is an extension of the C-tree decomposition algorithm.[8] The final D-tree decomposition method provides an effective way to decompose well- and under-constrained problems. The complexity of the algorithm is $O(n^4)$ where n is the number of the primitives in the constraint problem.

The main idea of the algorithms presented in this paper could be extended to cover all types of constraints since we use bipartite graphs. The algorithms could also be extended to the 3D case. We will explore these extensions in our future work. In this paper, we consider 2D problems with distance and angle constraints.

The rest of the paper is organized as follows. In Section 2, we introduce the basic concepts of constraint graph and the method to compute a maximum weighting of a constraint graph. In Section 3, we give a well-constrained completion algorithm of quadratic complexity. In Section 4, we propose the D-tree decomposition algorithm for under- and well-constrained problems. In Section 5, we present the conclusions.

2. Preliminary Notions and Algorithms

2.1. *Basic Concepts about Constraint Graphs*

In a geometric constraint problem, we consider two types of *geometric primitives*: points and lines in two dimensional Euclidean plane and two types of *geometric constraints*: the distance constraint between point/point, point/line and the angular constraint between line/line. In our algorithm, *positional constraints* will be added. These include to assign a coordinate to a point and the direction to a line. We will use p_i and l_i to represent points and lines respectively. The angular and distance constraints between two primitives o_1 and o_2 are denoted by $\mathrm{ANG}(o_1, o_2) = \alpha$ and $\mathrm{DIS}(o_1, o_2) = \delta$ respectively.

We use a bipartite graph $G = (V, C, E, \omega)$ or $G = (\mathbf{V}(G), \mathbf{C}(G), \mathbf{E}(G), \omega)$ to represent a geometric constraint problem, where V and C are the vertex sets of G, E is the edge set, ω defines a weight for each edge. The details are explained below.

(1) The vertices in V represent the geometric primitives, called *primitive vertices.* For every primitive vertex v in V, $\mathrm{DOF}(v)$ is the degree of freedom (abbr. DOF) of v, and $\mathrm{DOF}(V) = \Sigma_{v\in V}\mathrm{DOF}(v)$.
(2) The vertices in C represent the constraints, called *constraint vertices.* For $c \in C$, $\mathrm{DOF}(c)$ represents the DOF of c, that is, the number of scalar equations to represent the constraint, and $\mathrm{DOF}(C) = \Sigma_{c\in C}\mathrm{DOF}(c)$. Let H be a subgraph of G. A constraint vertex in H between two primitives in H is called an *internal constraint* of H. Otherwise, it is called an *external constraint* of H. We use $\mathrm{IDOF}(H)$ to denote the sum of the DOFs of the internal constraints of H.
(3) The edge set E is defined as: $E = \{(c, v))|v \in V, c \in C$, and c is a constraint involving $v\}$.
(4) For each edge $e \in E$, the *weight* $\omega(e)$ is a non-negative integer such that the sum of weights of all edges incident to a vertex o, denoted as $\omega(o)$, is no more than $\mathrm{DOF}(o)$. The initial weight of each edge is zero. A vertex o is said to be *saturated* or *unsaturated* if $\mathrm{DOF}(o) = \omega(o)$ or $\mathrm{DOF}(o) > \omega(o)$ holds respectively. A weighting of a graph is called a *maximal weighting* if the sum of all the weights of its edges is not less than the sum for any other weighting of that graph.
(5) From a weighting, we assign the edge directions. For an edge e between a primitive p and a constraint c, e is always directed from p to c. If $\omega(e) \neq 0$, e is also directed from c to p.

The bipartite graph G is called the *constraint graph* of the geometric constraint problem. Let U be a subset of $\mathbf{V}(G)$. The *subgraph induced* by U is the subgraph of G whose constraints are those constraints among primitives in U.

In this paper, the DOF of every geometric primitive is two. The DOF of the constraint $\mathrm{DIS}(p_1, p_2)$ is one if the distance is not zero; otherwise it is two. The DOF of constraint $\mathrm{DIS}(p, l)$ is one. The DOF of constraint $\mathrm{ANG}(l_1, l_2)$ is one. Figure 1 is a constraint problem and Figure 2 is its bipartite graph.

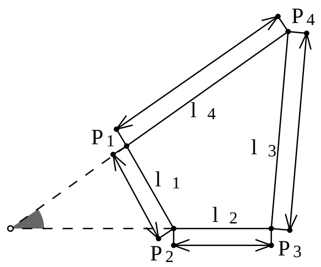

Fig. 1. A constraint problem: Lengths of four edges and $\mathrm{ANG}(l_2, l_4)$ are given

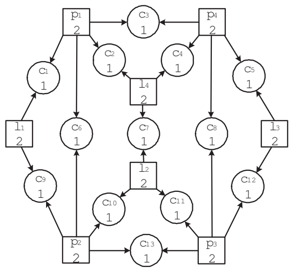

Fig. 2. Graph representation for the problem in Figure 1

A constraint graph G is called *(structurally) over-constrained* if there is a subgraph H of G such that H contains at least two primitive vertices and satisfies $\mathrm{IDOF}(H) > \mathrm{DOF}(\mathbf{V}(H)) - 3$. A constraint graph G is called *(structurally) well-constrained* if it is not over-constrained and $\mathrm{DOF}(\mathbf{C}(G)) = \mathrm{IDOF}(G) = \mathrm{DOF}(\mathbf{V}(G)) - 3$. A constraint graph G is called *(structurally) under-constrained* if G is not over-constrained and $\mathrm{DOF}(\mathbf{C}(G)) = \mathrm{IDOF}(G) < \mathrm{DOF}(\mathbf{V}(G)) - 3$.

A constraint system is called *geometrically well-constrained* if its shape has only a finite number of cases. In most cases, a structurally well-constrained problem is also geometrically well-constrained and hence defines a rigid body. But, in some special cases, a structurally well-constrained problem may have no solutions or an infinite number of solutions. To decide whether a constraint problem is geometrically well-constrained, we need to use techniques from symbolic computation.[9] In this paper, we will focus on the structure analysis of constrained problems and assume that a structurally well-constrained problem defines a rigid body.

2.2. *Computation of A Maximal Weighting*

A maximal weighting of a bipartite graph can be computed by incrementally improving *augmenting paths* which are paths directed from an unsaturated primitive to an unsaturated constraint.[2,10] Such paths are augmented by adding one to and subtracting one from the weights of alternate edges. The augmentation does not change the sum of weights of edges incident to any interior vertex of the path. It does change the sum of weights for end vertices but the weighting remains valid because the end vertices are unsaturated. For example, diagram (a) in Figure 3 is an augmenting path from p_1 to c_3 and diagram (b) is the augmented path.

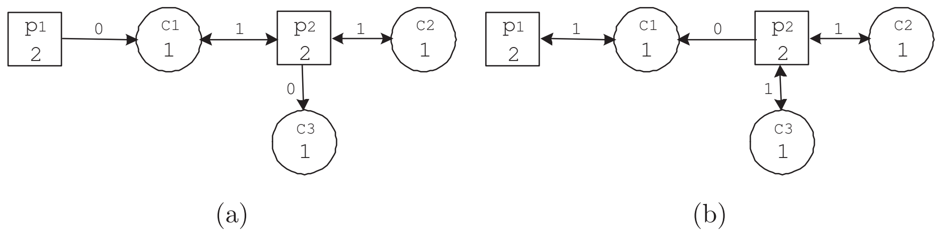

Fig. 3. Path augmentation

A weighting is maximal when there are no augmenting paths. Since each augmentation increases the sum of path weights by one, repeated augmentation converts an arbitrary weighting to a maximal weighting. After a constraint is added (or deleted), the augmentation efficiently computes the new maximal weighting from the old maximal weighting. In this case, augmenting paths are easier to find because they include the new constraint.[2]

There often exist more than one maximal weightings for a constraint graph. For example, in Figure 4, diagrams (a), (b) and (c) are three different maximal weightings for the constraint graph in Figure 1. Diagram (a) has unsaturated prim-

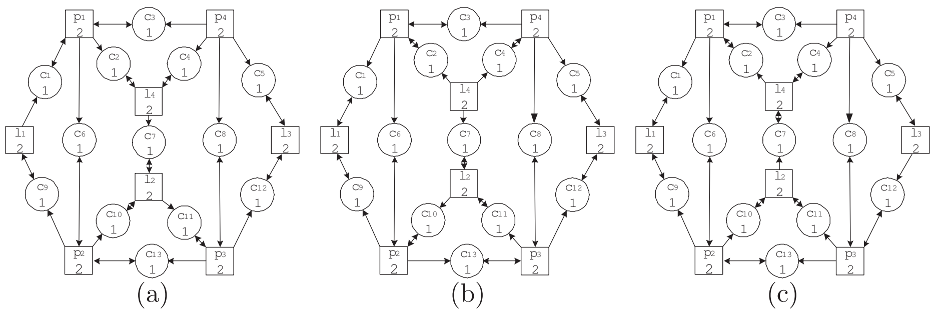

Fig. 4. Maximal weightings of the problem in Figure 1

itives p_4 and l_1. Diagram (b) has unsaturated primitives p_4 and l_4. Diagram (c) has unsaturated primitives p_4 and l_3. In Figure 4, the weight of each edge with single arrowhead is zero, and the weight of each edge with double arrowhead is one.

In a well-constrained or under-constrained graph with a maximal weighting, there exist no unsaturated constraints, but there might exist unsaturated primitives.

2.3. *Partition of a Constraint Graph*

After a maximal weighting of a graph is obtained, the constraint graph will be partitioned into disjoint subgraphs called *residual sets*. A set of vertices is *strongly connected* if there is a directed path from any vertex to any vertex in that set. A *residual set* is a set consisting of only one vertex or a strongly connected set which is not a proper subgraph of another strongly connected set. Residual sets could be computed with graph algorithms.[10]

For example, the constraint graph in Figure 4 (a) is partitioned into three residual sets: $\{p_4\}$, $\{p_1, c_1, c_3, l_1, c_9, p_2, c_6, c_{13}, p_3, c_8, c_{11}, l_2, c_7, c_{10}, l_4, c_2, c_4\}$, $\{l_3, c_5, c_{12}\}$. The constraint graph in Figure 4 (b) is partitioned into six residual sets: $\{l_4\}$, $\{p_4, c_4\}$, $\{p_1, c_2, c_3\}$, $\{p_2, c_6, c_{10}, p_3, c_{13}, c_8, l_2, c_7, c_{11}\}$, $\{l_3, c_5, c_{12}\}$, $\{l_1, c_1, c_9\}$. The constraint graph in Figure 4 (c) is partitioned into five residual sets: $\{p_4\}$, $\{l_3, c_5\}$, $\{p_3, c_8, c_{12}\}$, $\{p_1, c_2, c_3, p_2, c_6, c_{13}, l_2, c_{10}, c_{11}, l_4, c_4, c_7\}$, $\{l_1, c_1, c_9\}$.

3. Well-constrained Completion for Under-constrained Problems

In this section, we give an algorithm of quadratic complexity to solve Problem 1.1. The algorithms presented in this section will also be used by the D-tree decomposition algorithm in Section 4.

3.1. *Find Base Primitives and General Construction Sequences*

For a well-constrained or an under-constrained problem, we need to fix the position of some geometric primitives by adding some new constraints in order to determine the position of the diagram. For instance, to determine the position for a triangle whose three sides have known lengths, we need to fix the position of one of its vertices and the direction of one of its sides. These primitives are called *base primitives* and the newly added positional constraints are called *base constraints*. For a well-constrained problem G, after the base constraints are added, we have $\mathrm{DOF}(\mathbf{C}(G)) = \mathrm{DOF}(\mathbf{V}(G))$, and the new problem with these external constraints is called *strictly well-constrained*.

For well-constrained problems, we may find simpler solutions by selecting specific based primitives.[8] For under-constrained problems, the selection of base primitives is more complicated and more important than the well-constrained case as shown in the rest of this section.

We will try to find a *minimal rigid body* as shown in Figure 5. This constraint graph represents three types of constraints: $\mathrm{DIS}(p_1, p_2) = d$ $(d \neq 0)$, $\mathrm{DIS}(p, l) = d$, and $\mathrm{ANG}(l_1, l_2) = a$ $(a \neq 0)$. If such a minimal rigid body does not exist in the constraint problem, we will construct one by adding new primitives. Of course, the construction must not render the problem over-constrained.

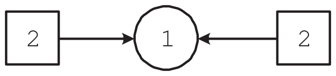

Fig. 5. Minimal rigid bodies

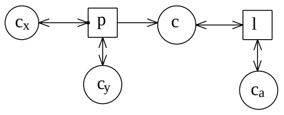

Fig. 6. Base primitives and constraints

Algorithm 3.1. The input is a graph G. The output is a set of base primitives and base constraints.

(1) If there is a distance constraint between a line l and a point p, we fix the position of p and the direction of l. The base primitives are p and l and the base constraints are assigning values to the coordinates of p and direction of l.
(2) If there is a nonzero distance constraint between two points p_1 and p_2, we fix the position of p_1 and the direction of the line passing through the two points.
(3) If there is a non zero angular constraint between two lines l_1 and l_2, we add the intersection p of l_1 and l_2 and fix the position of p and the direction of l_1.
(4) If there exists a line l, we add a new point p on line l and fix the position of p and the direction of l.
(5) If the conditions in 1-4 are not satisfied, all the geometric primitives must be free points. Let p_1 and p_2 be free points. We fix the position of p_1 and the direction of the line p_1p_2.

The output of Algorithm 3.1 is a constraint problem shown in Figure 6, where c_x, c_y, c_a are the newly added constraints meaning to fix the coordinates of a point and the direction of a line respectively. They are new types of constraints and are only used as base constraints. It is clear that $\{p, c_x, c_y\}$, $\{l, c, c_a\}$ are two residual sets. Similar ideas of pinning the problem were also used in Ref. 11.

Since each directed cycle of the constraint graph must lie entirely within a single residual set, and every edge incident to a residual set has a direction, we can impose a partial order among the residual sets.

Algorithm 3.2. The input is a set of base primitives S_b. The output is a sequence of residual sets with a partial order where the constraint subgraph induced by S_b is at the beginning of the sequence.

(1) Add the base primitives and constraints to the problem if they are not parts of the problem. Find the residual sets from a maximal weighting of the new constraint graph with methods in Section 2.
(2) Let S_b be the problem in Figure 6. Since c_x, c_y always direct to p and c_a always directs to l, we cannot change Figure 6 with a path augmentation. Therefore, $\{p, c_x, c_y\}$ and $\{l, c, c_a\}$ are two residual sets in the maximal weighting. We select $\{p, c_x, c_y\}$ as the first one and $\{l, c, c_a\}$ as the second one in the sequence. Delete all the edges whose tails are in S_b.
(3) Delete a residual set S which has no edges incident to it. This residual set is not connected with other parts of the graph. Repeat this step until no such residual set exists.
(4) If all the edges incident with S are directed from S to other residual sets, delete S and the edges whose tails are in S. Repeat this step until no residual sets exist.

Steps 3 and 4 could be improved with the incremental topological sort.[12]

The order of the deletion is the partial order among the residual sets. We thus obtain a sequence of residual sets

$$\mathcal{G} : S_1, S_2, \ldots, S_n,$$

such that $S_i \leq S_j$ for $i < j$. $\mathcal{G}$ is called a *general construction sequence* (abbr. GC) of the constraint problem. Geometrically, $S_i (i = 2, \ldots, n)$ will be determined by $S_1, \ldots, S_{i-1}$. Therefore, the largest $\mathrm{DOF}(\mathbf{V}(S_i))$ for $i = 1, \ldots, n$ is the maximal number of simultaneous equations to be solved in order to solve the GC. This number is called the *controlling DOF* of $\mathcal{G}$ and is denoted by $\mathrm{MDOF}(\mathcal{G})$.

For the problem in Figure 1, there are three essentially different GCs shown in Figure 4 before adding the base primitives:

$$\begin{aligned}
&\mathcal{G}_1 : \{p_4\}, \{p_1, c_1, c_3, l_1, c_9, p_2, c_6, c_{13}, p_3, c_8, c_{11}, l_2, c_7, c_{10}, l_4, c_2, c_4\}, \{l_3, c_5, c_{12}\} \qquad (1)\\
&\mathcal{G}_2 : \{l_4\}, \{p_4, c_4\}, \{p_1, c_2, c_3\}, \{p_2, c_6, c_{10}, p_3, c_{13}, c_8, l_2, c_7, c_{11}\}, \{l_3, c_5, c_{12}\}, \{l_1, c_1, c_9\}\\
&\mathcal{G}_3 : \{p_4\}, \{l_3, c_5\}, \{p_3, c_8, c_{12}\}, \{p_1, c_2, c_3, p_2, c_6, c_{13}, l_2, c_{10}, c_{11}, l_4, c_4, c_7\}, \{l_1, c_1, c_9\}.
\end{aligned}$$

If using $\mathcal{G}_1$, $\mathcal{G}_2$, and $\mathcal{G}_3$ to solve the problem, we have $\mathrm{MDOF}(\mathcal{G}_1) = \mathrm{DOF}(\{p_1, l_1, p_2, p_3, l_2, l_4\}) = 12$, $\mathrm{MDOF}(\mathcal{G}_2) = \mathrm{DOF}(\{p_2, p_3, l_2\}) = 6$, $\mathrm{MDOF}(\mathcal{G}_3) = \mathrm{DOF}(\{p_1, p_2, l_2, l_4\}) = 8$. It is clear that $\mathcal{G}_2$ is the best one and $\mathcal{G}_1$ is the worst one. If using Algorithm 3.1 to select base primitives, only $\mathcal{G}_2$ and $\mathcal{G}_3$

in (1) could be generated. In order to find better completions, we could find rigid bodies consisting of three geometric primitives and treat them as base primitives.[8] If using this heuristic, we will only generate $\mathcal{G}_2$ in Figure 4.

3.2. *Well-constrained Completion*

Let

$$S_1, S_2, \ldots, S_n \tag{2}$$

be a GC generated with Algorithm 3.2. Let $\mathcal{B}_k = \cup_{i=1}^{k} S_i$ and $\mathcal{U}_k = S_{k+1}$. We will call the problem of determining $\mathcal{U}_k$ based on $\mathcal{B}_k$ a *general basic merge pattern* (abbr. GBMP). If both $\mathcal{B}_k$ and $\mathcal{B}_k \cup \mathcal{U}_k$ are strictly well-constrained problems, then the GBMP is a basic merge pattern (BMP) introduced in Ref. 8.

The sum of DOFs for constraints between primitives in $\mathcal{B}_k$ and $\mathcal{U}_k$, denoted by $\mathrm{CN}(\mathcal{B}_k, \mathcal{U}_k)$, is called the *connection number*.

Let $\mathcal{U}_k = (\mathbf{V}(\mathcal{U}_k), \mathbf{C}(\mathcal{U}_k), \mathbf{E}(\mathcal{U}_k), \omega)$. Since $\mathrm{DOF}(\mathbf{V}(\mathcal{U}_k))$ constraints are needed to determine $\mathcal{U}_k$, we must have

$$\mathrm{CN}(\mathcal{B}_k, \mathcal{U}_k) + \mathrm{IDOF}(\mathcal{U}_k) \leq \mathrm{DOF}(\mathbf{V}(\mathcal{U}_k)). \tag{3}$$

Algorithm 3.3. The input is an under-constrained constraint graph G and the output is a well-constrained graph G' which contains G as a subgraph.

(1) Find a set of base primitives and add the base constraints with Algorithm 3.1.
(2) Find a GC (2) with Algorithm 3.2. Since the base primitives are always like Figure 6, we have $S_1 = \{p, c_x, c_y\}, S_2 = \{l, c, c_a\}$, and $S_1 \cup S_2$ is a strictly well-constrained problem.
(3) Let $k = 1$.
(4) If $k \geq n$, the algorithm terminates. Otherwise, Let $\mathcal{B}_k = \cup_{i=1}^{k} S_i$ and $\mathcal{U}_k = S_{k+1}$, $d_k = \mathrm{DOF}(\mathbf{V}(\mathcal{U}_k)) - \mathrm{CN}(\mathcal{B}_k, \mathcal{U}_k) - \mathrm{IDOF}(\mathcal{U}_k)$. Then by (3), $d_k \geq 0$. Also $\mathcal{B}_k$ is a strictly well-constrained problem.
(5) If $d_k = 0$, then $\mathcal{B}_k \cup \mathcal{U}_k$ is a strictly well constrained problem. Let $k := k + 1$ and goto Step 4.
(6) Now $d_k > 0$ and $\mathcal{B}_k \cup \mathcal{U}_k$ is under-constrained. Add a new constraint between $\mathcal{B}_k$ and $\mathcal{U}_k$ with Algorithm 3.4. In Algorithm 3.4, $\mathcal{U}_k = S_{k+1}$ is changed to a sequence of residual sets. Replace S_{k+1} in (2) with this sequence, and still denote the new sequence as (2). Goto Step 4.

In order to minimize the controlling DOF of the generated GCs, we adopt the following strategies when adding new constraints. Let $(\mathcal{B}, \mathcal{U})$ be a GBMP such that $\mathcal{B} \cup \mathcal{U}$ is under-constrained.

- Whenever possible, we will construct primitives in $\mathcal{U}$ one by one.
- We will add more angular constraints if possible, because constraint problems with more angular constraints are more likely to be solved explicitly.[8,13]

According to the above strategies, we design the following method for adding new constraints to an under-constrained problem.

Algorithm 3.4. The input is an under-constrained GBMP ($\mathcal{B}$,$\mathcal{U}$). The algorithm will add a new constraint to $\mathcal{U}$ and output a new GC: $\mathcal{B}, D_1, \ldots, D_s$ such that $\mathcal{U} = \cup_{i=1}^{s} D_i$.

(1) For a primitive u in $\mathcal{U}$, let EDOF(u) be the number of constraint vertices directed from $\mathcal{B}$ to u and IDOF(u) the number of constraint vertices directed from $\mathcal{U}$ to u. Since EDOF(u)+IDOF(u) $\leq$ DOF(u) $= 2$, EDOF(u) and IDOF(u) could be 0, 1 or 2. Search all the vertices in $\mathcal{U}$ to find a u with maximal EDOF(u). Note that $0 \leq d \leq 2$.
(2) The case EDOF(u) $= 2$ cannot happen. Because, in this case, u can be determined by $\mathcal{B}$ alone and u should be the only primitive in $\mathcal{U}$. This contradicts the condition that ($\mathcal{B}$,$\mathcal{U}$) is under-constrained.
(3) If EDOF(u) $= 1$, there exists an external constraint b_1 from $\mathcal{B}$ to u. We will add a new constraint from $\mathcal{B}$ to u to determine u.
    - Since IDOF(u) + EDOF(u) ≤ 2, we have IDOF(u) ≤ 1. If IDOF(u) $=$ 1, there is a constraint c directed from $\mathcal{U}$ to u. Since ($\mathcal{B}$,$\mathcal{U}$) is under-constrained, there exist unsaturated primitives in $\mathcal{U}$. Set the weight of the edge between u and c to zero. Constraint c becomes unsaturated. Do an augmentation for the augmenting path from an unsaturated primitive u_1 in $\mathcal{U}$ to c. We obtain a new maximal weighting $\mathcal{U}'$. The number of the constraints directed to u from $\mathcal{U}'$ will be 0.
    - Now IDOF(u) $= 0$ and EDOF(u) $= 1$. We use Algorithm 3.5 to add one new constraint b_2 from $\mathcal{B}$ directed to u. Now there are two constraints directed from $\mathcal{B}$ to u, which are denoted by $C_u = \{u_1, u_2\}$. This means that u can be determined by $\mathcal{B}$. With the terminology of constraint graphs, u and the constraint vertices directed from $\mathcal{B}$ to u consist of a strongly connected subgraph. So, we can let $D = \{u, C_u\}$. Partition $\mathcal{U}'$ into a sequence of residual sets $H_1 \leq \cdots \leq H_l$ with Algorithm 3.2 with D as the base primitives. Then $H_1 = D$, because there is no edge directed from $\mathcal{U}'$ to u. Return $\mathcal{B}, H_1, H_2, \ldots, H_l$.
(4) If EDOF(u) $= 0$, there exist no constraints between $\mathcal{B}$ and $\mathcal{U}$. Since $\mathcal{U}$ is under-constrained, there is an unsaturated primitive u in $\mathcal{U}$. Thus IDOF(u) ≤ 1. We could add one new constraint from $\mathcal{B}$ to u similar to Step 3 and return $\mathcal{B}$,$\mathcal{U}$.

Algorithm 3.5. The input is a GBMP ($\mathcal{B}$,$\mathcal{U}$) and an unsaturated primitive $u \in \mathcal{U}$. The output is a primitive $b \in \mathcal{B}$ and a constraint between u and b.

(1) If there exists a constraint between u and a primitive vertex in $\mathcal{B}$, denote this vertex as a. We need to consider two cases.
(2) If u is a point, we will select a point or a line $b \in \mathcal{B}$ such that $b \neq a$ and add a constraint DIS(b, u).

(3) Let u be a line. If there exist no angular constraints between $\mathcal{B}$ to $\mathcal{U}$, select a line $b \in \mathcal{B}$ such that $b \neq a$ and add a constraint ANG(b, u). In all other cases, select a point $b \in \mathcal{B}$ and add a constraint DIS(b, u). The motivation behind the selection of b is that we will try to add more angular constraints. But, if there exists an angular constraint between u and a primitive in $\mathcal{B}$, then we cannot add an angular constraint in order to avoid angular conflict.

Algorithm 3.3 solves the well-constrained completion problem. This algorithm improves the algorithm proposed in Ref. 2 in several aspects. First, by selecting base primitives carefully, we may generate better well-constrained completions. Second, as shown by the algorithm, all the newly added constraints are used to construct geometric primitives one by one. Third, we will try to add more angular constraints if possible. These strategies make the constraint completion easier.

Let n and e be the numbers of primitive and constraint vertices in G in Algorithm 3.3. Step 1 has complexity $O(e)$. In step 2, we need to find a GC, which has complexity $O(n(n + e))$.[8] Steps 3, 4 and 5 are either a constant or linear in terms of n and e. In Step 6, we need to call Algorithm 3.4. The complexity of Algorithm 3.4 is $O(m^2)$ $(m = |\mathcal{U}|)$ which is the cost to construct the residual sequence in the second part of Step 3. Let the GBMPs considered in Step 6 of Algorithm 3.3 be $(\mathcal{B}_i, \mathcal{U}_i)$ and m_i the number of primitive vertices in $\mathcal{U}_i$. Then $\sum_i m_i \leq n$. Therefore, the total complexity of the algorithm is $O((n + e)e) + O(\sum_i m_i^2) \leq O((n + e)e) + O(n^2) = O(ne + e^2 + n^2)$. In a well- or under-constrained problem, we have $e \leq 2n - 3$. Therefore, the complexity of the algorithm is $O(n^2)$.

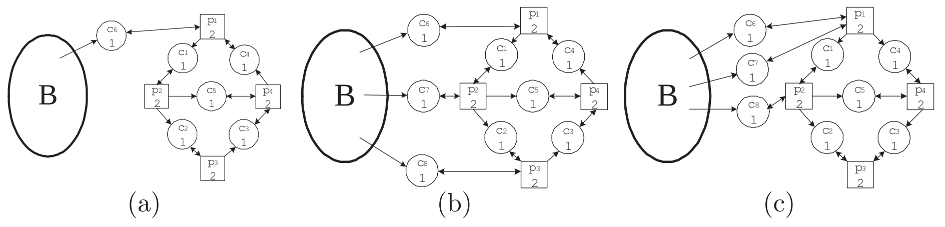

Fig. 7. An example of adding a new constraint

Diagram (a) in Figure 7 is a GBMP, where $\mathcal{U}$ consists of four points and six nonzero distance constraints. If we add new constraints c_7 and c_8 as shown in diagram (b), we need to solve a problem consisting in assembling two rigid bodies, which can be reduced to solve three linear equations and one equation of degree six.[8] But we can get an explicit construction sequence $\{p_1\}, \{p_2\}, \{p_4\}, \{p_3\}$ if we add new constraints c_7 and c_8 as shown in diagram (c) according to Algorithm 3.5. Let us explain Algorithm 3.4 with this example.

(1) Let $\mathcal{U} = \{p_1, c_4, c_6, p_2, c_1\, p_3, c_2, p_4, c_3, c_5\}$. In Step 1, we find only one u with

largest EDOF(u), which is p_1. We have IDOF(u) = 1.

(2) In Step 3, since p_3 is an unsaturated primitive, let the weight of the edge between p_1 and c_4 be zero. Thus c_4 is an unsaturate constraint vertex, and there is an augmenting path from p_3 to c_4. After the augmenting path is augmented, we obtain a new maximal weighting. Add a new external constraint c_7 between $\mathcal{B}$ and p_1 to obtain a new graph U'. Let $\mathcal{B}' = \mathcal{B} \cup \{p_1, c_6, c_7\}$. Find a new maximal weighting for $\mathcal{U}'$, which leads to a new GC: $\{p_1, c_6, c_7\}, \{p_2, c_1\}, \{p_4, c_4, c_5\}, \{p_3, c_2, c_3\}$. In the next step, p_2 will be constructed in a similar way.

Note that the well-constrained completion needs not to be unique. The addition of new constraints depend on the selection of base primitives in Algorithm 3.3, the selection of u in Step 1 of Algorithm 3.4, and the selection of b in Algorithm 3.5.

4. Decomposition for Under- and Well-constrained Problem

In this section, we give an algorithm which can be used to find a well-constrained completion for an under-constrained graph and to decompose the well-constrained completion into smaller parts which are represented by a tree, called D-tree. This D-tree decomposition algorithm will use the algorithms introduced in Section 3 on finding base primitives (3.1), finding GCs (3.2), and adding new constraints (3.4).

4.1. *A Decomposition Tree*

Let G be a well- or under-constrained graph and H a well-constrained subgraph of G. Let I be the subgraph of G induced by the set of $u \in \mathbf{V}(H)$ such that there exists at least one constraint between u and a vertex in $\mathbf{V}(G) \setminus \mathbf{V}(H)$. If $\mathbf{V}(I) \neq \mathbf{V}(H)$, H is called a *faithful subgraph* of G and I is called the *border*[14] of H.

The border I could be under-constrained. We use Algorithm 3.4 to generate a well-constrained completion for it and still denote it as I. This I is called a *cut subgraph* of G with respect to H. $\mathbf{V}(I)$ is a subset of $\mathbf{V}(H)$, where H is a rigid body. Suppose that H is already solved. Hence the values of the newly added constraints in I could be computed using the coordinates of the primitives in H.

The importance of faithful and cut subgraphs is that we can use them to reduce the original problem into two smaller ones.

Let H be a faithful subgraph of a graph G and I the corresponding cut subgraph. We construct a *split subgraph* S of G with respect to H as follows. Let $\mathbf{V}(S) = (\mathbf{V}(G) \setminus \mathbf{V}(H)) \cup \mathbf{V}(I)$. The subgraph induced by $\mathbf{V}(S)$ is called the *split subgraph* of G with respect to H.

Figure 8(a) is a constraint problem. Figure 8(b) is a faithful subgraph. The subgraph induced by $\{p_4, p_5, p_6\}$ is the cut subgraph. Figure 8(c) is the split subgraph. A new constraint c_{16} is added to make the split graph well-constrained.

A *decomposition tree* (abbr. D-tree) for a strictly well-constrained constraint graph G is a ternary tree. The root of the tree is G. For each node N in the tree,

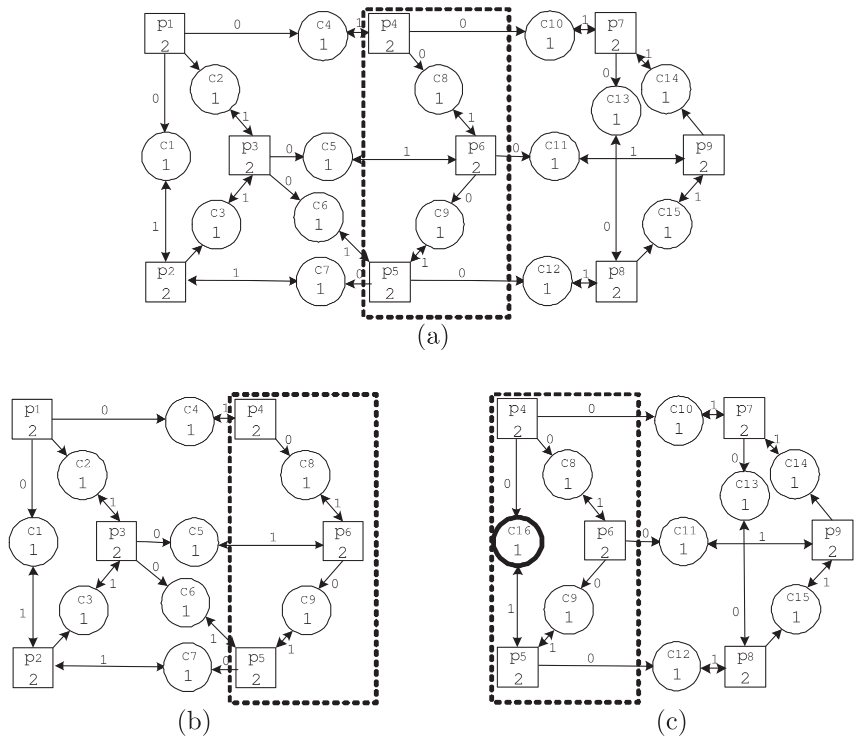

Fig. 8. An example to show the relation between the faithful and split subgraphs.

its left child L, middle child M and right child R are as follows.

(1) L is a faithful subgraph of N, M is the cut subgraph of N with respect to L, and R is the split subgraph of N with respect to L; or
(2) N is a strictly well-constrained problem and L is a GC for N, $M = \emptyset$ and $R = \emptyset$.

A D-tree for a constraint graph is defined to be a D-tree for a strictly well-constrained completion of the graph.

After a D-tree is obtained, we can use it to solve the problem as follows.

Algorithm 4.1. The input is a D-tree T. The outputs are the coordinates of the geometric primitives.

(1) We do a left to right depth-first search of the D-tree and consider three cases:
(2) The current node N is a GC $\mathcal{G}$ for a strictly well-constrained problem. The problem is reduced to the computation of $\mathcal{G}$ which is discussed in Ref. 8 and Ref. 13.
(3) The current node N has only a left child L. In this case, L is evaluated in Step 2 and N is solved.

(4) The current node N has three children. Due to the depth-first search procedure, we already solved the left child L. From L, we can compute the numerical values for the new constraints in M. With these new constraints, R is also a well-constrained problem. We can solve the right child recursively with this algorithm and merge the left and right children together to compute N. To merge L and R, we will consider the following cases.
   (a) First, since N is well-constrained, M cannot be the set of one primitive. Otherwise, the subgraph $M \cup R$ would be over-constrained.
   (b) If M consists of two parallel lines, the relative position of L and R cannot be fixed because there is a translational DOF between L and R. In this case, the problem is not a rigid body. Such an example is shown in Figure 9, where the constraints are $\mathrm{DIS}(p_1, l_2), \mathrm{DIS}(p_2, l_1), \mathrm{DIS}(p_3, l_1)$. Since $l_1 \parallel l_2$, the triangle $p_1p_2p_3$ can still move alone the direction of l_1.

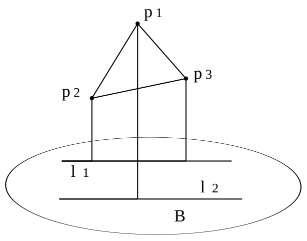

Fig. 9. I consists of two parallel lines

   (c) Otherwise, the relative position of L and R can be fixed and $L \cup R$ is a rigid body.

It is clear that the computation of a D-tree is reduced to the computation of GCs. We define the *controlling DOF of a D-tree T*, MDOF(T), to be the maximal MDOF(C) for all GCs C of T. A D-tree T for constraint graph G is called *minimal* if MDOF(T) is the smallest for all possible D-trees of G.

4.2. *An Algorithm to Find a D-tree*

Algorithm 4.2. The input is a well- or under-constrained graph G. The output is a D-tree for G.

Set $N = G$ as the initial value.

(1) Find a set of base primitives of N with Algorithm 3.1 and still denote the new graph with the base primitives and the base constraints as N.
(2) Find a GC for N:

$$\mathcal{G} = S_1, \ldots, S_n$$

such that the base primitives appear as the first residual set. Merge those adjacent S_i containing only one saturated primitive into one set to obtain a *reduced GC*:

$$\mathcal{G}' = S'_1, \ldots, S'_m, (m \leq n.)$$

(3) If $m = 1$, goto step 7. Otherwise, for $0 < k < m$, let $\mathcal{B}'_k = \cup_{i=1}^{k} S'_i$ and $\mathcal{U}'_k = S'_{k+1}$.

(4) If the problem is under constrained, that is, there is a k such that $\mathrm{CN}(\mathcal{B}_k, \mathcal{U}_k) + \mathrm{IDOF}(\mathcal{U}_k) < \mathrm{DOF}(\mathbf{V}(\mathcal{U}_k))$, then add one constraint with Algorithm 3.4 between $\mathcal{B}_k$ and $\mathcal{U}_k$. Still use the notations $\mathcal{G}, S_i, \mathcal{B}_i, \mathcal{U}_i$ for the new problem.

(5) Find a minimal k such that $\mathcal{B}'_k \cup \mathcal{U}'_k$ is a strictly well-constrained problem and there exists at least one primitive $o \in \mathcal{B}'_k$ such that there is no constraint between o and primitives in $\mathcal{U}'_k$. If such a k does not exist, goto the next step; otherwise goto step 8.

(6) There does not exist a faithful subgraph in $\mathcal{G}$.
  - If the problem is still under-constrained, goto Step 4.
  - Otherwise, select a new set of base primitives and goto Step 2 until there exist no new base primitives.

(7) We will solve N with $\mathcal{G}$. A D-tree for N is generated as follows: the left child of N is $\mathcal{G}$ and the middle and right children are empty sets. The algorithm terminates.

(8) $\mathcal{B}'_k$ induces a faithful subgraph $F = \mathcal{B}'_k$, a cut subgraph I and a split subgraph S. We build the D-tree as follows. The left child L of N is F; the middle child M of N is I; the right child R of N is S. Set $N := S$, remove the newly added constraints from N, and goto Step 1.

Let n and e be the numbers of primitive and edge vertices in G. Algorithm 4.2 has two main loops: loop one starting from Step 1 for each new N, loop two starting from Step 2 for each set of base primitives. After each loop one is executed, at least one primitive is constructed. So, the number of executions for loop one is at most n. The number of base primitives is $O(e)$. Therefore, the number of executions for loop two is at most $O(e)$. Similar to the complexity analysis for Algorithm 3.3, we can show that the complexity for each execution of loop two is $O(e^2 + n^2 + ne)$. Therefore, the total complexity of the algorithm is $O(ne(e^2 + n^2 + ne))$. In a well- or under-constrained problem, we have $e \leq 2n - 3$. Therefore, the complexity of the algorithm is $O(n^4)$.

Let G be the graph in Figure 10(a), which is also the root of the D-tree.

- In Step 1 we select $\{p_{13}, p_{14}\}$ as the base primitives.
- In Step 2, we generate a GC:
  $\mathcal{G}_0$: $\{p_{13}, p_{14}\}, \{p_1, p_2, p_3, p_4, p_{15}, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}, p_{16}, p_{17}, p_{18}\}$
  where the constraints are omitted.
- In Step 4, we add a constraint $d = \mathrm{DIS}(p_{14}, p_1)$ and generate a reduced GC: $\mathcal{G}_1$: $\{p_{13}, p_{14}, p_1, p_2, p_3, p_4, p_{15}\}, \{p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}, p_{16}, p_{17}, p_{18}\}$

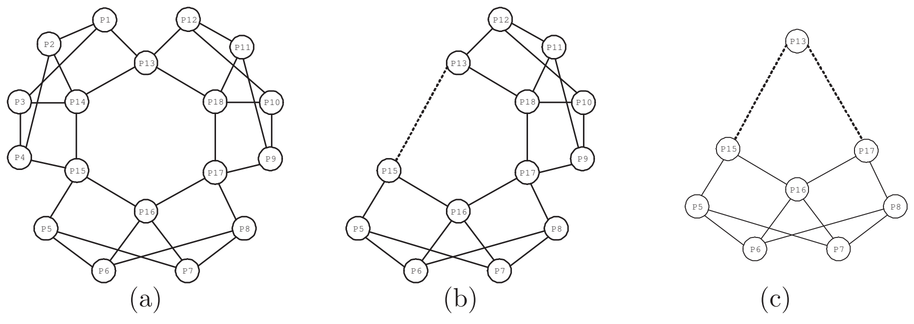

Fig. 10. Well-constrained completion for an under-constraint problem

- In Step 5, we find a $k = 1$. In Step 8, we generate a left child $\mathcal{G}_2$:

  $\mathcal{G}_2 : \{p_{13}, p_{14}, p_1, p_2, p_3, p_4, p_{15}\}$.

  The cut subgraph is $\{p_{13}, p_{15}\}$. A new constraint $\text{DIS}(p_{13}, p_{15})$ is added to make the cut subgraph a well-constrained one. The split subgraph is the problem in Figure 10 (b). Let N be the split subgraph. Go back to Step 1.
- Now N is the problem in Figure 10 (b). In Steps 1-4, select $\{p_{17}, p_{18}\}$ as the base primitives, add a new constraint $d = \text{DIS}(p_9, p_{18})$, and generate a new reduced GC:

  $\mathcal{G}_3 : \{p_{17}, p_{18}, p_9, p_{10}, p_{11}, p_{12}, p_{13}\}, \{p_5, p_6, p_7, p_8, p_{15}, p_{16}, p_{17}\}$.
- In Step 5, we find a $k = 1$. In Step 8, we generate a left child $\mathcal{G}_4$:

  $\mathcal{G}_4$: $\{p_{17}, p_{18}, p_9, p_{10}, p_{11}, p_{12}, p_{13}\}$.

  A constraint $\{p_{13}, p_{17}\}$ is added to make the cut subgraph well-constrained. The split subgraph is the problem in Figure 10 (c). Go back to Step 1.
- Now N is the problem in Figure 10 (c). Select $\{p_7, p_{16}\}$ as the base primitives. We add a new constraint $d = \text{DIS}(p_{16}, p_8)$ and generate a new GC:

  $\mathcal{G}_5$: $\{p_7, p_{16}, p_8, p_6, p_5, p_{15}, p_{17}, p_{13}\}$

  which is already an explicit construction sequence for problem N. Now the problem is reduced to the solving of three explicit construction sequences $\mathcal{G}_2, \mathcal{G}_4, \mathcal{G}_5$. The D-tree is given in Figure 11.

Figure 12 is an under-constrained problem where each line represents a distance constraint. This problem cannot be handled with the methods in Ref. 3 and Ref. 6, because its constraint graph is tri-connected. Using Algorithm 4.2, we select p_1 and p_2 as the base primitives and add a new constraint $\text{DIS}(p_1, p_3)$. Then the problem becomes well-constrained and has the following GC: $\{p_1\}, \{p_2\}, \{p_3\}, \{p_4\}, \{p_5, p_6, p_7, p_8\}$ where the constraint vertices are omitted.

5. Conclusions

Handling under-constrained problems is a key and difficult issue in GCS. It is also very useful, because it allows the CAD system to handle constraint problems incrementally. The main difficulty in handling under-constrained problems is how to change it into a well-constrained problem which is easy to solve, that is, how to find an optimal well-constrained completion.

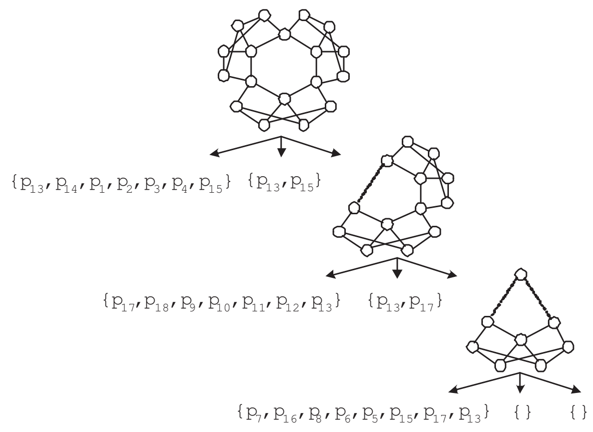
{p13, p14, p1, p2, p3, p4, p15}
{p13, p15}
{p17, p18, p9, p10, p11, p12, p13}
{p13, p17}
{p7, p16, p8, p6, p5, p15, p17, p13}
{ }
{ }

Fig. 11. A D-tree for the problem in Figure 10.

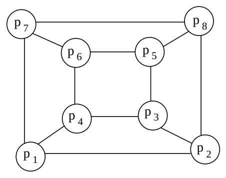

Fig. 12. A tri-connected under-constraint problem

In this paper, we proposed several strategies and developed algorithms which provide a partial solution to the optimal well-constrained completion problem. The key idea is to use base primitives, GCs, constructing single primitives, and D-tree decompositions. The final D-tree decomposition method provides an effective way to decompose well- and under-constrained problems.

Acknowledgements

This paper is partially supported by a National Key Basic Research Project of China. The authors want to thank the anonymous referees for valuable suggestions.

References

1. R. Joan-Arinyo, A. Soto-Riera and S. Vila-Marta, Tools to Deal with Under-constrained Geometric Constraint Graphs, Workshop on Geometric Constraint Solving 2003, http://www.mmrc.iss.ac.cn/˜ ascm/ascm03/gcsweb/robert.pdf.

2. R.S. Latham and A.E. Middleditch, Connectivity Analysis: A Tool for Processing Geometric Constraints, *Comput. Aided Des.* **28** (1996) 917-928.
3. I. Fudos and C.M. Hoffmann, A Graph-Constructive Approach to Solving Systems of Geometric Constraints, *ACM Trans. Graph.* **16**(2) (1997) 179-216.
4. B. Yuan, Y. Yuan, S.M. Hu and J.G. Sun, Strategy on Underconstrained Parametric Design (in Chinese), *Chinese J. of Adv. Soft. Res.* **6**(4) (1999) 305-310.
5. K.Y. Lee, O.H. Kwon, J.Y. Lee and T.W. Kim, A Hybrid Approach to Geometric Constraint Solving with Graph Analysis and Reduction, *Adv. Eng. Softw.* **34** (2003) 103-113.
6. 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, *Proc. ACM SM03* (ACM Press, New York, 2003) pp. 33-44.
7. G. Trombettoni and M. Wilczkowiak, GPDOF: a Polynomial Algorithm to Decompose Under Constrained Systems: Applications to 3D Model Construction, this volume.
8. X.S. Gao, Q. Lin and G.F. Zhang, A C-tree Decomposition Algorithm for 2D and 3D Geometric Constraint Solving, *Comput. Aided Des.* **38** (2006) 1-13.
9. X.S. Gao and S.C. Chou, Solving Geometric Constraint Systems, II. A Symbolic Computational Approach, *Comput. Aided Des.* **30** (1998) 115-122.
10. M. Gondran and M. Minoux, *Graphs and Algorithms*, translated by Steven Vajda. (John Wiley & Sons, New York, 1984).
11. H. Lamure and D. Michelucci, Qualitative Study of Geometric Constraints, in *Geometric Constraint Solving and Applications* (Springer, Berlin, 1998) 234-258.
12. R. Hoover, *Increamental Graph Evaluation*, thesis, Cornell University, Ithaca (1987).
13. X.S. Gao, D. Lei, Q. Liao and G. Zhang, Generalized Stewart-Gough Platforms and their Direct Kinematics, *IEEE Trans. Robot.* **21**(2) (2005) 141-151.
14. J.F. Dufourd, P. Mathis and P. Schreck, Geometric Construction by Assembling Solved Subfigures, *Artif. Intell.* **99** (1998) 73-119.