A C-Tree Decomposition Algorithm for 2D and 3D Geometric Constraint Solving

Xiao-Shan Gao, Qiang Lin, Gui-Fang Zhang

A C-tree decomposition algorithm for 2D and 3D geometric constraint solving☆

Xiao-Shan Gao[a,*], Qiang Lin, Gui-Fang Zhang[b]

[a]*Key Laboratory of Mathematics Mechanization, Institute of Systems Science, AMSS, Academia Sinica, Beijing 100080, China*
[b]*Department of Computer Science and Technology, Tsinghua University, Beijing, China*

Received 14 June 2004; received in revised form 4 March 2005; accepted 10 March 2005

Abstract

In this paper, we propose a method which can be used to decompose a 2D or 3D constraint problem into a C-tree. With this decomposition, a geometric constraint problem can be reduced into basic merge patterns, which are the smallest problems we need to solve in order to solve the original problem in certain sense. Based on the C-tree decomposition algorithm, we implemented a software package MMP/Geometer. Experimental results show that MMP/Geometer finds the smallest decomposition for all the testing examples efficiently.

Keywords: Geometric constraint solving; Parametric CAD; General construction sequence; Basic merge pattern; Decomposition tree; Graph algorithm

1. Introduction

Geometric constraint solving (GCS) is one of the key techniques in 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 molecular modelling, robotics and computer vision. There are four major approaches to GCS: the numerical approach [14,26,31], the symbolic approach [15,23,25,33], the rule-based approach [2,24,34,35] and the graph-based approach [6,7,12,17,20,28,30]. This paper will focus on using graph algorithms to decompose large constraint problems into smaller ones.

In [32], Owen proposed a GCS method based on the tri-connected decomposition of graphs, which may be used to reduce a class of constraint problems into constraint problems consisting of three primitives. In [7,16], Hoffmann et al. proposed a method based on cluster formation to solve 2D and 3D constraint problems. An algorithm was introduced by Joan-Arinyo et al. in [21] to decompose a 2D constraint problem into an s-tree. This method is equivalent to the methods of Owen and Hoffmann, but is conceptually simpler.

The above approaches use special constraint problems, i.e. triangles, as basic patterns to solve geometric constraint problems. In [28], Latham and Middleditch proposed a connectivity analysis algorithm which could be used to decompose a constraint problem into what we called the general construction sequence (defined in Section 2). A similar method based on maximal matching of bipartite graphs was proposed by Lamure and Michelucci [27]. In [17], Hoffmann et al. gave an algorithm to find rigid bodies in a constraint problem. Based on this, a general approach to GCS was proposed [18]. In [19], Jermann et al. also gave a general approach to GCS based on the idea in [17].

In this paper, we propose a method which can be used to decompose a general 2D or 3D constraint problem into a C-tree (connectivity tree). The algorithm is inspired by two facts. First, the general construction sequence obtained with Latham–Middleditch's algorithm reduces the original constraint problem into smaller ones. But, in many cases these smaller problems could be further simplified. Second, we observed that not all rigid bodies in a constraint problem can be used to split the original problem. We introduced the key concept of *faithful subgraph*, which may lead to a split of the constraint problem. The C-tree decomposition algorithm

☆ Partially supported by a National Key Basic Research Project of China (2004CB318000) and by a USA NSF grant CCR-0201253.

* Corresponding author. Tel.: +81 10 6254 1831; fax: +81 10 6263 0706.

E-mail address: xgao@mmrc.iss.ac.cn (X.-S. Gao).

consists of two main steps: using the general construction sequence to find faithful subgraphs and using the faithful subgraph to split the constraint problem into two subproblems. The complexity of the algorithm is $O(n^2(n+e)e)$, where n and e are the numbers of geometric objects and constraints, respectively. Major advantage of the algorithm is that it can be used to decompose a general constraint problem into certain kind of smallest problems and it leads to a simple and efficient implementation.

A C-tree is a binary tree. For each node in the tree, its left child is a rigid body which will be solved first. After the left child is solved, we may use the information from the left child to solve the right child and to merge the left and right children to solve the constraint problem represented by the node. All leaves of the C-tree are general construction sequences. Therefore, solution of a constraint problem is reduced to the solution of general construction sequences with a C-tree decomposition.

We show that the solution of a general construction sequence can be reduced to the solution of *basic merge patterns*, which are the smallest problems we need to solve in order to solve the original problem in certain sense. We give a classification of the basic merge patterns both in 2D and 3D cases and show that some of the basic merge patterns have closed-form solutions.

We say that a graph decomposition method for GCS is a general method if it can be used to handle all constraint problems. Among the general GCS methods [18,19,24,27,28], the method MFA proposed in [18] and the C-tree method can be used to find a smallest decomposition in certain sense. The MFA and C-tree methods have the same complexity. Both can be used to solve 2D and 3D problems, although paper [18] focuses on the 2D case and this paper focuses on both 2D and 3D cases. Comparing to the algorithm in [18], our algorithm is simpler and easy to implement.

Based on the C-tree decomposition algorithm and several other algorithms proposed by us, we implemented a GCS software package MMP/Geometer in the Windows environment with VC++. Experimental results show that the software package finds the smallest decomposition for all the testing examples efficiently.

The rest of the paper is organized as follows. In Section 2, we introduce the methods to generate general construction sequences. In Section 3, we give the algorithm to generate the C-tree. In Section 4, we give a classification of the basic merge patterns. In Section 5, we report the experimental results for our implementation of the C-tree decomposition algorithm. In Section 6, we present the conclusion.

2. General construction sequence

2.1. Basic concepts

In the two-dimensional Euclidean plane, we consider two types of *geometric primitives*: points and lines and two types of *geometric constraints*: the distance constraint between point/point, point/line and the angular constraint between line/line. In the three-dimensional Euclidean space, we consider three types of geometric primitives: points, planes and lines and two types of geometric constraints: the distance constraints between point/point, point/line, point/plane, line/line and the angular constraints between line/line, line/plane, plane/plane. A *geometric constraint problem* consists of a set of geometric primitives and a set of geometric constraints among these primitives. 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 will use p_i, h_i and l_i to represent points, planes and lines, respectively.

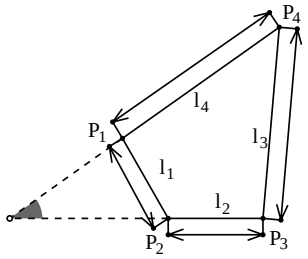

Fig. 1. A 2D problem: lengths of four edges and ANG (l_2,l_4) are given.

We use a *constraint graph* to represent a constraint problem. The vertices of the graph represent the geometric primitives and the edges represent the constraints. For a constraint graph G, we use $\mathbf{V}(G)$ and $\mathbf{E}(G)$ to denote its sets of vertices and edges, respectively. Fig. 2 is the graph representation for the constraint problem in Fig. 1.

For an edge e in a constraint graph, let $\mathrm{DOC}(e)$ be the valence of e, which is the number of scalar equations required to define the constraint represented by e. Most constraints considered by us have valence 1. There are several exceptions: (1) Constraint $\mathrm{DIS}(p_1,p_2)=0$. In this case, $p_1=p_2$. In 2D case, the constraint has valence 2; in 3D, the constraint has valence 3. We assume that this case does no occur. (2) Constraint $\mathrm{DIS}(p_1,l_1)=0$ has valence 2 in 3D. (3) Constraint ANG $(h_1,h_2)=0$ has valence 2 in 3D. (4) Constraint ANG $(l_1,l_2)=0$ has valence 2 in 3D. These constraints are *degenerate cases*.

For a geometric primitive o, let $\mathrm{DOF}(o)$ be the degrees of freedom for o, which is the number of independent parameters required to determine the geometric

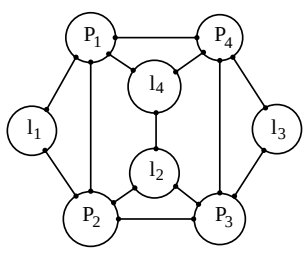

Fig. 2. Graph representation for the problem in Fig. 1.

primitive. For a constraint graph G, let $\mathrm{DOF}(G)=\sum_{v\in\mathbf{V}(G)}\mathrm{DOF}(v)$, $\mathrm{DOC}(G)=\sum_{e\in\mathbf{E}(G)}\mathrm{DOC}(e)$. In the constraint graphs, we use n lines to represent a constraint of valency n.

Let $R=3$ in 2D and $R=6$ in 3D. A constraint graph G is called *structurally well-constrained* if $\mathrm{DOC}(G)=\mathrm{DOF}(G)-R$ and for every subgraph H of G, $\mathrm{DOC}(H)\leq\mathrm{DOF}(H)-R$. A constraint graph G is called *structurally over-constrained* if there is a subgraph H of G satisfying $\mathrm{DOC}(H)>\mathrm{DOF}(H)-R$. A constraint graph G is called *structurally under-constrained* if G is not over-constraint and $\mathrm{DOC}(G)<\mathrm{DOF}(G)-R$.

A constraint system is called geometrically well-constrained if its shape has only a finite number of cases. In most cases, a constraint problem represented by a structurally well-constrained graph is geometrically well-constrained and hence defines a rigid body. But, in some special cases a constraint problem represented by a structurally well-constrained graph may have no solutions or an infinite number of solutions. In this paper, we will concern the structure solvability of the constraint problem only. Therefore, when we say *rigid bodies* in this paper, we mean structurally well-constrained problems.

2.2. General construction sequence

In a geometric constraint problem, a general construction sequence (abbr. GC) is a sequence:

$$\mathcal{C}:\ C_1, C_2, \ldots, C_n$$

where each C_i is a set of geometric primitives, such that

1. The subgraph induced by $\mathcal{B}_i=\cup_{k=1}^{i}C_k$ is well-constrained for each $1\leq i\leq n$. Therefore, we may assume that B_i is a rigid body.
2. No proper subsets of C_i satisfy condition 1.

If each C_i contains only one primitive, we call the corresponding GC an *explicit construction sequence*.

The largest $\mathrm{DOF}(C_i)$ for $i=1,\ldots,n$ is the maximal number of simultaneous equations to be solved in order to solve the above GC. This number is called the *controlling degree of freedom* of $\mathcal{C}$ and is denoted by $\mathrm{MDOF}(\mathcal{C})$.

For the example in Fig. 1, there are two essentially different GCs:

$$\begin{aligned}\mathcal{G}_1:&\ \{p_1\},\{p_4\},\{l_4\},\{l_2,p_2,p_3\},\{l_1\},\{l_3\}\\ \mathcal{G}_2:&\ \{p_1\},\{p_2\},\{p_4,p_3,l_2,l_4\},\{l_1\},\{l_3\}\end{aligned}\qquad(1)$$

We have $\mathrm{MDOF}(\mathcal{G}_1)=\mathrm{DOF}(\{l_2,p_2,p_3\})=6$, $\mathrm{MDOF}(\mathcal{G}_2)=\mathrm{DOF}(\{p_4,p_3,l_2,l_4\})=8$. It is clear that $\mathcal{G}_1$ is better than $\mathcal{G}_2$.

In [28], Latham and Middleditch proposed an algorithm which may be used to reduce a well-constrained problem into a GC. Their method is based on the maximal b-matching from graph theory which is of complexity $O(n(e+n))$, where n is the number of vertices and e is the number of edges in the constraint graph [1].

2.3. Find base primitives

In order to generate GCs with nice properties, we will add three more constraints to a set of primitives in 2D and six more constraints to a set of primitives in 3D before using the algorithm mentioned in Section 2.2. These primitives are called *base primitives* and will be generated firstly in the GC. The geometric meaning of this step is as follows: a rigid body in the plane has three DOFs and a rigid body in the space has six DOFs. By fixing the position of the base primitives, we can find the absolute position of the rigid body. After this step, a structurally well-constrained problem G will satisfy the condition $\mathrm{DOC}(G)=\mathrm{DOF}(G)$, which is called *strictly well-constrained*.

In the 2D case, finding base primitives is easy. Since the constraint problem is a rigid body, there exists at least one distance constraint. We may use the two primitives involved in this constraint as the base primitives and add three more constraints as follows:

- If the two base primitives are two points, we may fix the position of one point and the direction of the line passing through the two points.
- If the two base primitives are a point p and a line l, we may fix the position of p and the direction of l.

If using the above method to select base primitives, both $\mathcal{G}_1$ and $\mathcal{G}_2$ in (1) could be generated. As a heuristic, we will try to find rigid bodies consisting of three geometric primitives and treat them as base primitives. If using this heuristic, then we will only generate $\mathcal{G}_1$. In 2D case, we need to find the graphs listed in Fig. 3, which can be done with the following algorithm.

Algorithm 2.1. The input is a well-constrained constraint graph. The output is a set of three vertices such that there exist constraints between each pair of them and at least one of the constraints is a distance constraint.

S1 Search all the edges $e=(o_1,o_2)$ representing a distance constraint such that one of o_1, o_2 is a point.
S2 For each e, search all the vertices o having a constraint with o_1.
S3 If o_2 has a constraint with o, then return o_1, o_2, o. The algorithm terminates.

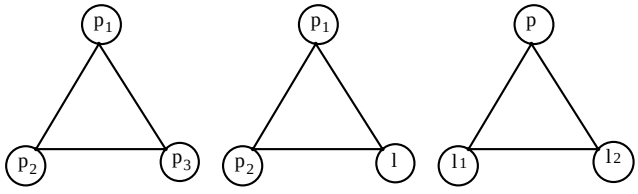

Fig. 3. Rigid bodies with three primitives in 2D case.

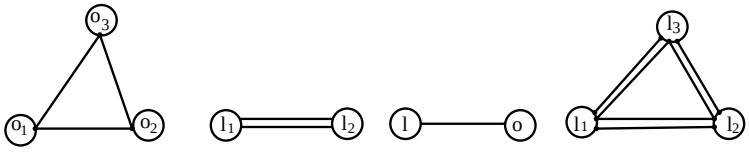

Fig. 4. Rigid bodies with two or three primitives in 3D case.

Let e be the number of edges $l_1,\ldots,l_e$ in the graph G, and d_i the number of constraints involving l_i. Then the main loop in Step 1 will execute e times. The loop in Step 2 for an edge l_i will execute d_i times. If using an adjacent matrix to represent the graph, Step 3 needs only one operation. Therefore, the complexity of the algorithm is:

$$\sum_{i=1}^{e}\sum_{j=1}^{d_i} \mathrm{O}(1) = \sum_{i=1}^{e} \mathrm{O}(d_i) = \mathrm{O}\left(\sum_{i=1}^{e} d_i\right) = \mathrm{O}(e)$$

The last step is true because $\sum_{i=1}^{e} d_i = 2e$, since each constraint involving two primitives.

To find base primitives for a 3D constraint problem, we first try to find a rigid body consisting of two or three primitives in the constraint problem. The four graphs in Fig. 4 represent such rigid bodies in 3D. The first diagram in Fig. 4 represents three cases: three points, two points and one plane, one point and two planes. The third diagram in Fig. 4 represents two cases: a line and a point or a line and a plane.

The following algorithm tries to find a set of base primitives by first finding a rigid body with two or three primitives. If such rigid bodies do not exist, we will select a distance constraint and use the primitives involved in this constraint as the base primitives. The reason is that a rigid body always contains at least one distance constraint.

Algorithm 2.2. The input is a well-constrained constraint graph for a 3D problem. The output is a set base primitives.

S1 Find the first and fourth diagram in Fig. 4 with Algorithm 2.1. If such a sub-graph is found, then add six constraints to fix the position of the rigid body and return the three primitives as base primitives. For instance, if the three primitives are three points p_1, p_2, p_3, we may add the six constraints as follows: $p_1=(0,0,0)$ is fixed as the origin; $p_2=(d_1,0,0)$ is a point on the x-axis; $p_3=(x,y,0)$ is a point on the xy-plane.

S2 Search all the edges $e=(o_1,o_2)$ representing a distance constraint $\mathrm{DIS}(o_1,o_2)=d$.

S3 If o_1 is a line and o_2 is a point, we may add six constraints as follows. Fix o_1 to be the z-axis and $o_2=(d,0,0)$ a point on the x-axis.

S4 If both o_1 and o_2 are lines, we may add six constraints as follows. Add a new point p which is on o_2 and has a distance d to o_1. Then we may select o_1 and p as base primitives and add six constraints similar as case S3.

S5 Now o_1, o_2 must be points or planes. Since there is a distance constraint between them, one of them must be a point. Let o_2 be a point. Since a constraint involving a point must be a distance constraint, we may find a third primitive o_3 such that there is a constraint $\mathrm{DIS}(o_2,o_3)=d'$. The three primitives o_1, o_2, o_3 could have three possibilities.

S6 If both o_1 and o_3 are points, we may add the following six constraints: $o_2=(0,0,0)$, $o_1=(d,0,0)$, $o_3=(x,y,0)$.

S7 If o_1 is a point and o_3 is a plane, add six constraints as follows. Take o_3 to be the xy-plane; $o_2=(0,0,d')$ a point on the z-axis; and o_1 a point in the xz-plane.

S8 If both o_1 and o_3 are planes, we may add a new point p which is the foot of the perpendicular line drawn from o_2 to o_1 and treat the problem similar as case S7.

Let us look at the constraint problem in Fig. 5(a), where each line represents a distance constraint between two points. For this problem, there are three essentially different GCs $\mathcal{C}_1$, $\mathcal{C}_2$, $\mathcal{C}_3$:

$$C_1:\ \{P\},\{Q\},\{A\},\{B\},\{C\},\{D\},\{U,V,W\}$$

$$C_2:\ \{P\},\{Q\},\{U\},\{V\},\{W\},\{A,B,C,D\}$$

$$C_3:\ \{W\},\{D\},\{P,Q,A,B,C,D,U,V\}$$

It is clear that the GCs depend on the base primitives. The base primitives for GCs $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ are $\{P,Q,A\}$, $\{P,Q,U\}$, $\{W,D,C\}$, respectively. Actually, if using Algorithm 2.2 to select base primitives, only $\mathcal{C}_1$ and $\mathcal{C}_2$ could be generated.

We have $\mathrm{MDOF}(\mathcal{C}_1)=\mathrm{DOF}(\{U,V,W\})=9$, $\mathrm{MDOF}(\mathcal{C}_2)=\mathrm{DOF}(\{A,B,C,D\})=12$, $\mathrm{MDOF}(\mathcal{C}_3)=\mathrm{DOF}(\{P,Q,A,B,C,U,V\})=21$. It is clear that $\mathcal{C}_1$ is the best GC. To solve this problem directly without decomposition, we need to solve 21 quadratic equations. Using C_1, we need only to

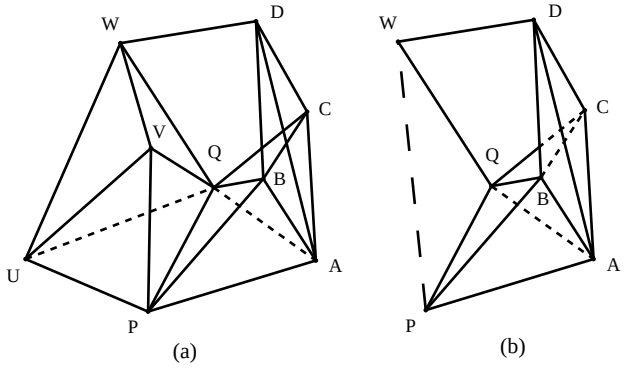

Fig. 5. A 3D constraint problem.

solve nine quadratic equations. In the next section, we will show that the problem can be further simplified to solve one quadratic equation.

3. A C-tree decomposition algorithm

3.1. A general decomposition tree: C-tree

In order to define the C-tree, we introduce the concept of deficit, which is a generalization of the deficit function defined in [21].

The *deficit* of a geometric constraint graph G is defined as

$$\text{deficit}(G) = \text{Dof}(\mathbf{V}(G)) - \text{Doc}(\mathbf{E}(G)) - R$$

where $R=3$ or 6 in 2D or 3D cases, respectively. If G is a structurally well-constrained problem then deficit $(G)=0$. If G is not structurally over-constrained then deficit $(G)\geq 0$.

Theorem 3.1. *Let G be a structurally well-constrained graph, H, I, S subgraphs of G such that $\mathbf{V}(I)=\mathbf{V}(H)\cap\mathbf{V}(S)$ and there exist no constraints between vertices in $\mathbf{V}(H)-\mathbf{V}(I)$ and $\mathbf{V}(S)-\mathbf{V}(I)$ (Fig. 6). Then we have deficit $(H)+$ deficit $(S)=$ deficit (I). If H is also structurally well-constrained, we have deficit $(S)=$ deficit (I).*

Proof. Since $\mathbf{V}(I)=\mathbf{V}(H)\cap\mathbf{V}(S)$ and there exist no constraints between vertices in $\mathbf{V}(H)-\mathbf{V}(I)$ and $\mathbf{V}(S)-\mathbf{V}(I)$, we have $\text{DOF}(\mathbf{V}(H))+\text{DOF}(\mathbf{V}(S))-\text{DOF}(\mathbf{V}(I))=\text{DOF}(\mathbf{V}(G))$ and $\text{DOC}(\mathbf{E}(H))+\text{DOC}(\mathbf{E}(S))-\text{DOC}(\mathbf{E}(I))=\text{DOC}(\mathbf{E}(G))$. Thus deficit $(H)+$deficit $(S)-$deficit $(I)=\text{DOF}(\mathbf{V}(H))+\text{DOF}(\mathbf{V}(S))-\text{DOF}(\mathbf{V}(I))-(\text{DOC}(\mathbf{E}(H))+\text{DOC}(\mathbf{E}(S))-\text{DOC}(\mathbf{E}(I)))-R=\text{DOF}(\mathbf{V}(G))-\text{DOC}(\mathbf{E}(G))-R=0$. The last equation is due to the fact that G is structurally well-constrained. □

Let G be a structurally well-constrained graph and H a structurally well-constrained subgraph of G. Let I be the set of vertices $u\in H$ such that there exists at least one constraint between u and a vertex in $\mathbf{V}(G)-\mathbf{V}(H)$. If $I\neq\mathbf{V}(H)$, H is called a *faithful subgraph*. The importance of a faithful subgraph is that we can use it to reduce the original problem into two smaller ones.

Let H be a faithful subgraph of G. We may construct a *split subgraph* S of G with H as follows: $\mathbf{V}(S)=(\mathbf{V}(G)-\mathbf{V}(H))\cup I$, where I is defined in the preceding paragraph. If S is structurally well-constrained, S is the split subgraph. Otherwise, by Theorem 3.1, deficit $(I)=$deficit (S). Then we may add deficit (I) *auxiliary constraints* between vertices in I to make the new graph S structurally well-constrained.

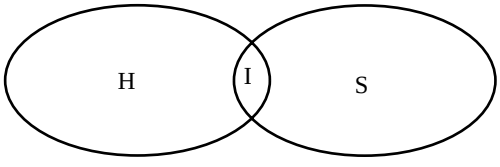

Fig. 6. The relation between the faithful and split subgraphs.

This can be done with the algorithm in [28]. The algorithms in [7,22] can also be used to obtain a structurally well-constrained problem in certain cases. This new graph S is called the split subgraph.

For instance, let G be the graph in Fig. 5(a), H the subgraph of G induced by $\{W,U,V,P,Q\}$. Then H is a faithful subgraph of G, because $I=\{W,P,Q\}\neq\mathbf{V}(H)$. The split subgraph S is the one in Fig. 5(b), where the constraint between W/P is the auxiliary constraint. The geometric meaning is as follows. We first solve the constraint problem H, which is a rigid body. To solve the remaining part S, we need to add an auxiliary constraint between W/P to make S a well-constraint problem. This is possible because, we may solve the problem represented by H and we may use the information from H to determine the distance between W/P. Then the solution of G is reduced to the solution of two smaller problems H and S.

Definition 3.2. A C-tree for a constraint graph G is a binary tree. The root of the tree is G. For each node N in the tree, its left child L and right child R are as follows:

1. L is a faithful subgraph of N and R is the split subgraph of N with L; or
2. L is a GC for N and $R=\varnothing$.

All leaves are GCs.

Continue with the example in Fig. 5(a). $H=\{W,U,V,P,Q\}$ is a faithful subgraph. The split subgraph S is the one in Fig. 5(b). Then H and S are the left and right children of G in the C-tree in Fig. 7. The left children for H and S are their GCs, respectively.

We may say that the C-tree is a natural generalization for the s-tree from [21]. In an s-tree, when a problem is divided into two smaller problems P_1 and P_2, P_1 and P_2 always have two common primitives. In a C-tree, P_1 and P_2 could have any number of common primitives.

After a C-tree is obtained, we may use it to solve the constraint problem as follows.

Algorithm 3.3. The input is a C-tree T. The outputs are the coordinates of the geometric primitives:

S1 We do a left to right depth-first search of the C-tree and consider three cases: S2, S3, or S4.

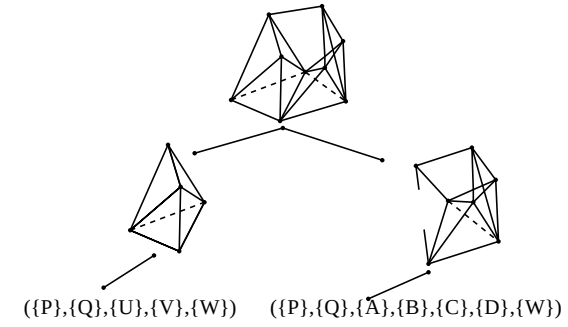

Fig. 7. A C-tree for the problem in Fig. 5.

S2 The current node N is a GC. The problem is reduced to the computation of a GC which will be discussed in Section 4.

S3 The current node N only has the left child L, which is a *GC*. In this case, L is evaluated and N is solved.

S4 The current node N has two children. Due to the depth-first search procedure, we already solved the left child L which is a rigid body. From L, we may compute the numerical values for the auxiliary constraints in the right child R. Now the right child becomes a structurally well-constrained problem. We may solve the right child recursively with this algorithm and merge the left and right children together to get the information about N.

Note that the merging of the left and right children is easy, since they are connected by sharing several geometric primitives. We omit the details and illustrate the procedure with an example. If L and R share three points p_1, p_2, p_3, a merging process is given below. Let $p_{1,i}$ and $p_{2,i}$ be the corresponding points of p_i in L and R. We may fix R and move L to a correct position as follows. First, do a translation $p_{2,1}$–$p_{1,1}$ to L so that L and R will share the same point p_1. Second, do a rotational around point p_1 to L so that L and R will share the same point p_2. Third, do a rotation around line p_1p_2 to L so that L and R will share point p_3. Now the relative position of L and R are fixed.

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

To solve the problem in Fig. 5(a) with the C-tree in Fig. 7, we first compute the left child of the root using the explicit construction sequence $\{P\}$, $\{Q\}$, $\{U\}$, $\{V\}$, $\{W\}$. Then, we may compute the distance between P/W and solve the right child using the explicit construction sequence $\{P\}$, $\{Q\}$, $\{A\}$, $\{C\}$, $\{D\}$, $\{W\}$. In this way, to solve the problem in Fig. 5, we need only to find the intersection of three spheres, which can be reduced to the solution of one quadratic equation.

3.2. An algorithm to find a C-tree

Algorithm 3.4. The input is a structurally well-constrained graph G. The output is a C-tree for G.

Let $T=G$ as the initial value.

S1 Select a set of base primitives with Algorithms 2.1 and 2.2. After three (2D case) or six (3D case) new constraints are added, we obtain a new graph H from T.

S2 With the algorithms in [28], we may find a GC for H

$$\mathcal{C}: C_1, \ldots, C_m$$

such that the base primitives will always appear first in the GC. Using Proposition 4.3 to decide whether $\mathcal{C}$ is angular conflict. If it is, the problem generally has no solutions and the algorithm terminates.

S3 If $|C_i|=1$ for some i, it is relatively easy to compute C_i. We merge all the neighboring C_i containing only one primitive into one set to obtain a *reduced GC*:

$$\mathcal{C}': C'_1, \ldots, C'_s.$$

S4 If $s=1$, then H can be solved by explicit constructions and $\mathcal{C}$ is a construction sequence for H. We may generate a C-tree from $\mathcal{C}$ as follows: the left child of T is $\mathcal{C}$ and the right child is the empty set. The algorithm terminates.

S5 Let $\mathcal{B}'_i=\cup_{j=1}^{i} C'_j$ and k the smallest number satisfying the following condition. There exists at least one primitive $o\in\mathcal{B}'_k$ such that there are no constraints between o and primitives in C'_i, $i=k+1,\ldots,s$. If such a k does not exist, let $k=s$.

S6 If $k=s$, there exist no faithful subgraphs in this GC. Find a set of new base primitives for T to generate a new H as done in S1 and go to S2. If no new base primitives exist, we have to solve G with the GC $\mathcal{C}$. We may generate a C-tree for T as follows: the left child of T is $\mathcal{C}$ and the right child is the empty set. The algorithm terminates.

S7 Otherwise, $k<s$. Now $\mathcal{B}'_k$ induces a faithful subgraph F. We build the C-tree as follows. The left child of T is F. The left child of F is the GC: $C_1,\ldots,C_d$ where d is an integer satisfying $\cup_{j=1}^{d} C_j=\mathcal{B}'_k$; the right child of F is the empty set. The right child of T is the split subgraph G' of G with F. Set $T=G'$ go to S1.

Let n and e be the numbers of vertices and edges in G. As mentioned in Section 2.2, Step S1 has complexity O(e). In S2, we need to use the maximal b-matching from graph theory to find a GC, which has complexity O($n(n+e)$) [1]. Steps S3–S5 are linear in terms of n and e. Therefore, S2 is the controlling step for the loop started at step S6. At the worst case, the loop started by S6 could run for O(e) times, since the number of base primitives is linear in terms of e. The loop started at S7 could run n times. Therefore, the total complexity of the algorithm is O($n^2(n+e)e$).

We may modify Algorithm 3.4 to find the minimal C-tree for G by searching all the possible base primitive sets in Step S1. For a given set of base primitives, the generated GC is unique due to the fact that the corresponding strong connected sets in the graph decomposition is unique [28]. Therefore, by searching all the possible base primitives, we have obtained all the GCs and thus all the possible C-trees for the problem.

3.3. Working examples

Let G be the graph in Fig. 5(a), which is also the root of the C-tree. In Step S1 of Algorithm 3.4, we select P, Q, U as the base primitives. In other words, we will construct G

starting from these points. In S2, we generate the following GC:

$$\mathcal{C}: \{P\},\{Q\},\{U\},\{V\},\{W\},\{A,B,C,D\}$$

In S3, the single points in $\mathcal{C}$ are collected together to form the following reduced GC:

$$\mathcal{C}': \{P,Q,U,V,W\},\{A,B,C,D\}$$

Step S4 does nothing. In S5, $k=1$ since we may choose $o=U$. In S6, $k\neq s=2$ so nothing is done. This means that $\mathcal{C}_1'=\{P,Q,U,V,W\}$ is a faithful subgraph. In S7, new notes are added to the C-tree. The left child of the root is $\mathcal{C}_1'$ and the left child of $\mathcal{C}_1'$ is the following construction sequence:

$$\mathcal{C}_4: \{P\},\{Q\},\{U\},\{V\},\{W\}$$

The right child of the root is the split subgraph of H by $\mathcal{C}_1'$, which is the graph in Fig. 5(b). Details on how to generate the split subgraph is given in Section 3.1. Now, we may repeat the above process for the right child, which can be generated with the following GC:

$$\mathcal{C}_5: \{P\},\{Q\},\{A\},\{B\},\{C\},\{D\},\{W\}$$

Now the C-tree in Fig. 7 is generated. Basically speaking, to solve the problem, we need to solve two GCs: $\mathcal{C}_4$ and $\mathcal{C}_5$. Since $\mathrm{MDOF}(C_4)=\mathrm{MDOF}(C_5)=3$, which is the simplest case we could have.

Fig. 8 is a more difficult constraint problem, where each edge represents a distance between two points. Fig. 9 is the C-tree for it. In Algorithm 3.4, we select points A, B, C as the base primitives. Latham–Middleditch's algorithm will give a GC as follows:

$$\mathcal{D}_1: \{A\},\{B\},\{C\},\{D\},\{E,F,G,H\},\{I,L,J,K\}$$

The reduced GC in S3 is:

$$\mathcal{D}_1: \{A,B,C,D\},\{E,F,G,H\},\{I,L,J,K\}$$

In S5, $k=2$, which will lead to the children of the root (Fig. 9).

Note that the problem in Fig. 5(a) could be solved with the cluster merging method proposed in [16]. But the problem in Fig. 8 cannot be simplified with the cluster formation method.

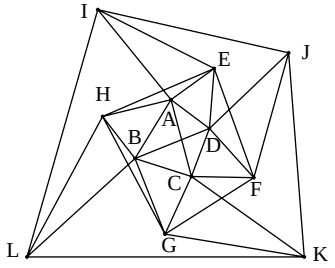

Fig. 8. A 3D constraint problem about 12 points.

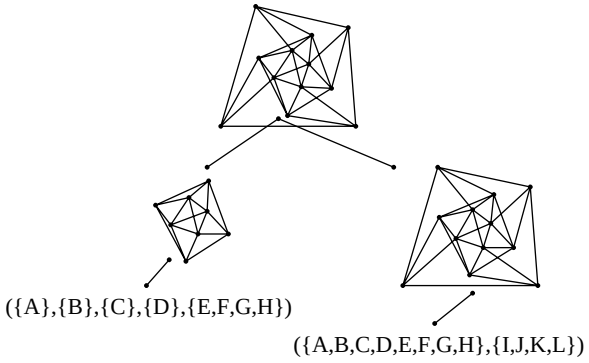

Fig. 9. C-tree for the constraint problem in Fig. 8.

4. Basic merge patterns

With a C-tree decomposition, solving of a constraint problem is reduced to the solving of a GC. In this section, we will show how to solve a GC. Suppose that we want to solve a constraint problem represented by a GC:

$$\mathcal{C}: C_1, C_2, \ldots, C_n \tag{2}$$

All we need to do is to determine C_i based on $C_1,\ldots,C_{i-1}$. Let:

$$\mathcal{B}=\bigcup_{k=1}^{i-1} C_k, \quad \mathcal{U}=C_i$$

We call the problem of determining $\mathcal{U}$ based on $\mathcal{B}$ *basic merging pattern*. $\mathcal{U}$ and $\mathcal{B}$ are called *the dependent objects* and *the base objects*, respectively. From the definition of GCs, a basic merge pattern $(\mathcal{B},\mathcal{U})$ has the following properties:

1. $\mathcal{B}$ and $(\mathcal{B}\cup\mathcal{U})$ are rigid bodies.
2. There is no subset V of $\mathcal{U}$ such that $(\mathcal{B}\cup V)$ is a rigid body.

The sum of $\mathrm{DOC}(e)$ for all edges e between $\mathcal{B}$ and $\mathcal{U}$, denoted by $\mathrm{CN}(\mathcal{B},\mathcal{U})$, describes an important natural of the merging step, and is called *the connection number*. Since both $\mathcal{B}$ and $(\mathcal{B}\cup\mathcal{U})$ are rigid bodies, we need exactly $\mathrm{DOF}(\mathcal{U})$ constraints to determine $\mathcal{U}$. In other words, we have

$$\mathrm{CN}(\mathcal{B},\mathcal{U})+\mathrm{DOC}(\mathcal{U})=\mathrm{DOF}(\mathcal{U}) \tag{3}$$

4.1. Classification of 2D basic merge patterns

Theorem 4.1. *In a 2D constraint problem, we have:* $2\leq \mathrm{CN}(\mathcal{B},\mathcal{U})\leq|\mathcal{U}|$

Proof. Since $\mathcal{U}$ contains at least one element and $\mathcal{B}$ is a rigid body, $\mathrm{CN}(\mathcal{B},\mathcal{U})\geq 2$. If $|\mathcal{U}|>1$ then for every vertex v in $\mathcal{U}$ there exists at least one constraint between v and another vertex in V. Otherwise, v can be determined by $\mathcal{B}$ alone, which contradicts to the minimum property of $\mathcal{U}$.

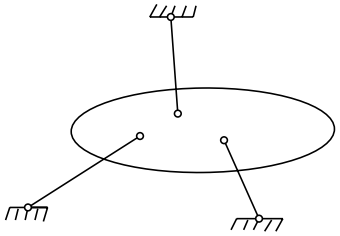

Fig. 10. The 2D general Stewart platform.

Then, there exist at least $(2\times|\mathcal{U}|)/2=|\mathcal{U}|$ constraints between primitives in U, i.e. $\mathrm{DOC}(\mathcal{U})\geq|\mathcal{U}|$. We have $\mathrm{DOC}(\mathcal{U})=2|\mathcal{U}|$ since all the geometric primitives are of DOF two. By (3), $\mathrm{CN}(\mathcal{B},\mathcal{U})=\mathrm{DOF}(\mathcal{U})-\mathrm{DOC}(u)\leq 2|\mathcal{U}|-|\mathcal{U}|=|\mathcal{U}|$. □

Based on $\mathrm{CN}(\mathcal{B},\mathcal{U})$, the solution of a basic merge pattern $(\mathcal{B},\mathcal{U})$ can be divided into the following three cases:

1. If $\mathrm{CN}(\mathcal{B},\mathcal{U})=2$, $\mathcal{U}$ consists of one geometric element o, and o can be constructed explicitly.
2. If $\mathrm{CN}(\mathcal{B},\mathcal{U})=3$, by (3), $\mathrm{DOC}(\mathcal{U})=\mathrm{DOF}(\mathcal{U})-3$. Hence $\mathcal{U}$ is a rigid body and can be solved independently of $\mathcal{B}$. To solve this basic merge pattern, we may first solve $\mathcal{U}$ and then assemble two rigid bodies $\mathcal{B}$ and $\mathcal{U}$ according to three constraints.

This case deserves special attention. A basic merge pattern $(\mathcal{B},\mathcal{U})$ satisfies condition $\mathrm{CN}(\mathcal{B},\mathcal{U})=3$ is called a *generalized Stewart platform* (abbr. GSP). Detailed description of GSP may be found in Section 4.2. Fig. 10 is the illustration of a 2D GSP. The problem is to determine the position of $\mathcal{U}$ assuming that (1) the position of $\mathcal{B}$ is known; (2) $\mathcal{U}$ has been solved, that is, the relative positions of the geometric primitives in $\mathcal{U}$ are known; and (3) the values of the three constraints between $\mathcal{B}$ and $\mathcal{U}$ are given. Closed-form solutions to the 2D GSP have been found [11].

3. If $\mathrm{CN}(\mathcal{B},\mathcal{U})>3$, the problem becomes more complicated. Now $\mathcal{U}$ is not a rigid body anymore. We need to use the constraints inside $\mathcal{U}$ and those between $\mathcal{U}$ and $\mathcal{B}$ to determine $\mathcal{U}$. We will use the numerical computational method proposed in [14] to solve these kinds of problems.

As an example, the basic merging patterns for the case $|\mathcal{U}|=5$ are shown in Fig. 11. In these diagrams, we use circles to represent the vertices with two degrees of freedom, circles labelled **R** to represent the rigid bodies, and the thin lines to represent the constraints between $\mathcal{B}$ and $\mathcal{U}$.

In Fig. 11, $\mathcal{U}$ is treated as a set of geometric elements. We may further to decompose $\mathcal{U}$ as a set of rigid bodies if possible. This will simplify the problem greatly. In Fig. 12, the six merging patterns for case $|\mathcal{U}|=5$ in Fig. 11 are simplified to three patterns. Actually, these patterns represent more cases, because the rigid bodies in these patterns may be of any size. The first two cases are used in [29] as basic patterns to solve constraint problems.

4.2. Classification of 3D basic merge patterns

Theorem 4.2. *In a 3D constraint problem, let V_3 be the set of points and planes on the dependent object $\mathcal{U}$, V_4 the set of lines on $\mathcal{U}$. We have* $3\leq\mathrm{CN}(\mathcal{B},\mathcal{U})\leq\mathrm{DOF}(\mathcal{U})-|\mathcal{U}|=2|V_3|+3|V_4$

Proof. Since $\mathcal{U}$ contains at least one geometric primitive and $\mathcal{B}\cup\mathcal{U}$ is a rigid body, $\mathrm{CN}(\mathcal{B},\mathcal{U})$ should be greater than or equal to the degree of freedom for one primitive. Hence $\mathrm{CN}(\mathcal{B},\mathcal{U})\geq 3$. From [28], $\mathcal{U}$ can be changed to a strongly connected directed graph. Since a strongly connected graph with n vertices has at least n edges, $\mathcal{U}$ contains at least $|\mathcal{U}|$ constraints, i.e. $\mathrm{DOC}(\mathcal{U})\geq|\mathcal{U}|$. Since both $\mathcal{B}$ and $\mathcal{B}\cup\mathcal{U}$ are rigid bodies, we need exactly $\mathrm{DOF}(\mathcal{U})=3|V_3|+4|V_4|$ constraints to determine $\mathcal{U}$. In other words, we have

$$\mathrm{CN}(\mathcal{B},\mathcal{U})+\mathrm{DOC}(V)=\mathrm{DOF}(\mathcal{U})=3|V_3|+4|V_4|$$

Thus $\mathrm{CN}(\mathcal{B},\mathcal{U})\leq\mathrm{DOF}(\mathcal{U})-|\mathcal{U}|=2|V_3|+3|V_4|$. □

Based on $\mathrm{CN}(\mathcal{B},\mathcal{U})$, the solution of a basic merge pattern $(\mathcal{B},\mathcal{U})$ can be divided into the following cases:

1. If $\mathrm{CN}(\mathcal{B},\mathcal{U})=3$, $\mathcal{U}$ consists of a point or a plane, which can be constructed explicitly.
2. If $\mathrm{CN}(\mathcal{B},\mathcal{U})=4$, $\mathcal{U}$ consists of a line, which can be constructed explicitly.
3. If $\mathrm{CN}(\mathcal{B},\mathcal{U})=5$, $\mathcal{U}$ consists of a line l and several points on l. Suppose that there are m points p_i, $i=1,\ldots,m$ on l. After renaming the points, $\mathrm{DIS}(p_i,p_{i+1})$, $i=1,\ldots,n-1$ must be known. Otherwise, $\mathrm{DOF}(\mathcal{U})>5$ which is contradict to the fact that $\mathrm{CN}(\mathcal{B},\mathcal{U})=5$.

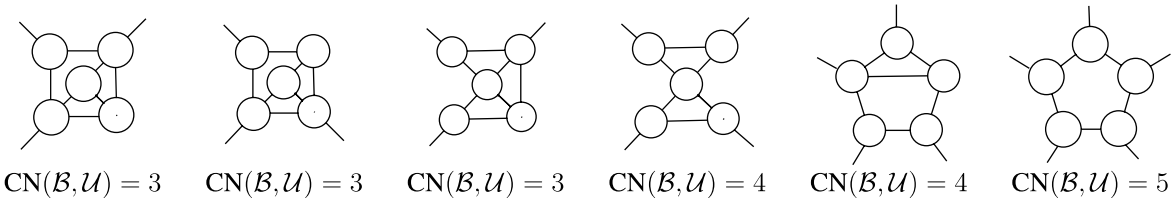

Fig. 11. Basic merging patterns consisting of five primitives.

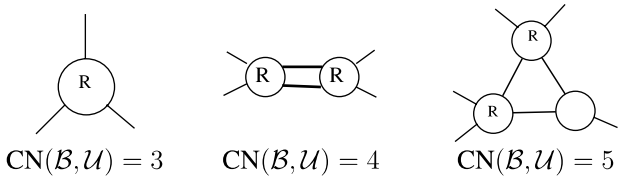

Fig. 12. Basic merge patterns for rigid bodies.

4. If $CN(\mathcal{B},\mathcal{U})=6$, by (3), $DOC(\mathcal{U})=DOF(\mathcal{U})-6$. Hence $\mathcal{U}$ is a rigid body and can be solved independently of $\mathcal{B}$. To solve this basic merge pattern, we may first solve $\mathcal{U}$ and then assemble two rigid bodies $\mathcal{B}$ and $\mathcal{U}$ according to six constraints.
5. If $CN(\mathcal{B},\mathcal{U})>6$, the problem becomes more complicated. Now $\mathcal{U}$ is not a rigid body anymore. We need to use the constraints inside $\mathcal{U}$ and those between $\mathcal{U}$ and $\mathcal{B}$ to determine $\mathcal{U}$. Techniques from AI could be used to simplify some of the cases [5]. Generally, we will use the numerical computational method proposed in [14] to solve these kinds of problems.

Let $(\mathcal{B},\mathcal{U})$ be a basic merging pattern such that $CN(\mathcal{B},\mathcal{U})=6$. Then both $\mathcal{B}$ and $\mathcal{U}$ are rigid bodies. Hence, it may be considered as an *assembly problem*. We need to assemble two rigid bodies according to six constraints. This problem can be divided into four cases:

3D3A There are three distance and three angular constraints between $\mathcal{B}$ and $\mathcal{U}$
4D2A There are four distance and two angular constraints between $\mathcal{B}$ and $\mathcal{U}$
5D1A There are five distance and one angular constraints between $\mathcal{B}$ and $\mathcal{U}$
6D All the six constraints between $\mathcal{B}$ and $\mathcal{U}$ are distance constraints

We cannot have more than three angular constraints due to the fact that a 3D rigid body only need three angular constraints to determine its directions.

This case deserves special attention because it is closely related to the famous *Stewart Platform* [4], which is a 6D basic merge pattern where all distance constraints are between points. This platform is extensively studied

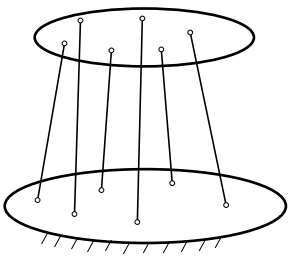

Fig. 13. The 3D GSP.

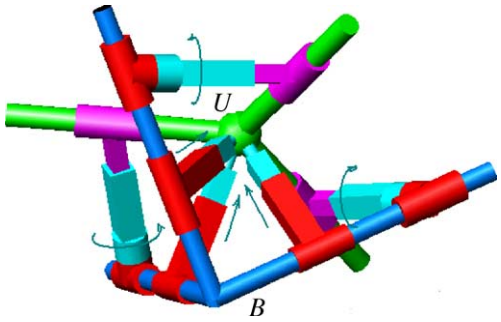

Fig. 14. A 3D3A GSP.

because it has many important applications. For a survey, please consult [4]. Most of the work on Stewart platform is focused on the *forward displacement* problem: for a given position of $\mathcal{B}$ and a set of values of the distances, to determine the position of $\mathcal{U}$. This is exactly what we are trying to do in solving a basic merge pattern.

The system $(\mathcal{B},\mathcal{U})$ satisfying $CN(\mathcal{B},\mathcal{U})=6$ will be called a *generalized Stewart platform* (abbr. GSP). Fig. 13 is an illustration of a GSP in 3D. Fig. 14 is a 3D3A GSP, where $\mathcal{B}$ and $\mathcal{U}$ (the three lines perpendicular to each other) are connected with three rotational and three distance constraints. In [10], we have given the upper bounds for the number of solutions for all GSPs and closed-form solutions for the 3D3A GSPs.

4.3. Detection of angular conflict

In the above, we only concern the structure of the basic merge patterns. It could happen that a structurally well-constrained problem may have no solutions. One such case is to have too many angular constraints, which can be easily detected. If one of the following cases occurs in a basic merge pattern $(\mathcal{B},\mathcal{U})$, we say that it is an *angular conflict pattern*.

1. In 2D case, a line in $\mathcal{U}$ has more than one angular constraints with elements in $\mathcal{B}$. In 3D case, a plane or a line in $\mathcal{U}$ has more than two angular constraints with elements in $\mathcal{B}$.
2. In 2D case, if $CN(\mathcal{B},\mathcal{U})=3$ and there are more than one angular constraints between $\mathcal{B}$ and $\mathcal{U}$. If $CN(\mathcal{B},\mathcal{U})=3$, both $\mathcal{B}$ and $\mathcal{U}$ are rigid bodies and we need only one angular constraint to determine the rotational degree of freedom of $\mathcal{U}$. In 3D case, if $CN(\mathcal{B},\mathcal{U})=6$ and there are more than three angular constraints between $\mathcal{B}$ and $\mathcal{U}$.
3. In 2D case, let l be the number of lines in $\mathcal{U}$. The number for the angular constraint between $\mathcal{U}$ and $\mathcal{B}$ and between primitives in $\mathcal{U}$ is more than l. In 3D case, let l and h be the numbers of lines and planes in $\mathcal{U}$. The number for the angular constraints between $\mathcal{U}$ and $\mathcal{B}$ and between primitives in $\mathcal{U}$ is more than $2(l+h)$.

A GC involving an angular conflict pattern is called an *angular conflict GC*.

Proposition 4.3. *In general, an angular conflict pattern cannot be realized in the Euclidean space*.

Proof. Let us consider the 2D case. Since a line and a rigid body both have one angular (directional) DOF, it is clear that the first two cases in the definition of an angular conflict pattern will lead to angular conflicts and hence cannot be realized in the Euclidean space. In the third case, we may first consider those lines which have angular constraints with lines in $\mathcal{B}$. By case 1, each line can have only one such constraint and the direction of this line is determined. Let us call them fixed lines. In a similar way, we may consider lines with angular constraints with fixed lines. Repeat the process, we may assume that there are t lines left and there exist more than t angular constraints between themselves. But, t lines have at most t directional DOFs. The more than t angular constraints will lead to angular conflicts. □

5. Implementation and experimental results

In this section, we report a software package based on the C-tree decomposition algorithm. The GCS algorithms implemented in the software are for 2D and 3D. The current interface is only for 2D.

5.1. A General framework of GCS

We first give a general framework of GCS, which will be used in our software package. This general framework consists of four major steps.

1. Use the algorithm LIM0 in [8] to find explicit construction sequences for the problem. This simple algorithm is of linear complexity and could be used to solve about 80% of the 512 geometry problems in [3].
2. If the LIM0 algorithm fails, we use the geometric transformation method to find explicit construction sequences [9]. This is a quadratic algorithm and is complete for drawing constraint problems of simple polygons.
3. If the above step fails, we use the C-tree decomposition algorithm to reduce the problem into basic merge patterns, which are the smallest problem we have to solve in order to solve the original problem. This process is illustrated in Fig. 15.
4. There are three types of basic merge patterns. The explicit construction means to construct one geometric primitive, which is generally easy. The GSP is to determine the position of one rigid body, which has closed-form solution in 2D [11] and certain cases in 3D [10]. For the general type, we usually do not have closed-form solutions. For the problems without closed-form solutions, we use the numerical method reported in [14] to solve the problem. This method can solve equation systems consisting of 100 quadratic equations in seconds.

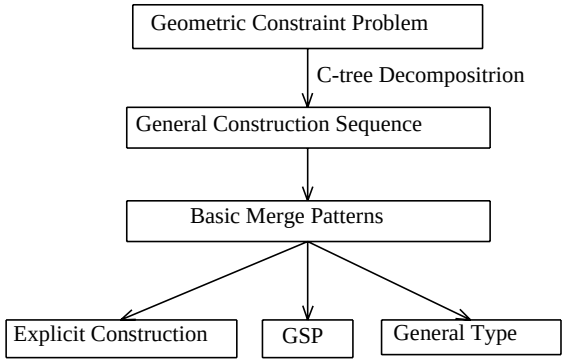

Fig. 15. Solving a constraint problem.

5.2. The software and experimental results

We have implemented the general framework of GCS introduced in Section 5.1 in our software package MMP/Geometer [13] MMP/Geometer is implemented in Windows environment with VC++. Although the general framework of GCS can be used to both 2D and 3D problems, the current version of MMP/Geometer only handles 2D problems. This is mainly due to the difficulty to implement a 3D sketch interface.

After the algorithm is selected, we still need to add details in the implementation to enhance the performance of the algorithm. One problem is the ambiguities rising from operations like 'intersection of two circles' or 'intersection of a circle and a line'. These constructions have several solutions. When the user changes a constraint value, the program will compute the position of all points in the figure automatically. Our goal is to keep a continuous movement of

Table 1
Running times and final GCs for problems in Figs. 16 and 17

Figure	Time in second	Final GC_S in the C-tree
(a)	0.537	$\{P_4,P_3,L_3,L_2,P_2,P_1,L_4,L_1\}$, $\{P_7,P_6,L_6,L_5,P_5,L_7,P_8\}$, $\{(P_4,P_3,L_2), (L_7,L_6,P_5)\}$
(b)	0.513	$\{A,B,C,D,E,F\}$; $\{G,H,I,J\}$; $\{(A,D,F), (G,H,I)\}$
(c)	0.464	$\{A,B,C,D,E,F\}$; $\{B,F,G,H,I,J,K,L\}$
(d)	0.347	$\{(A,B,C), (D,E,F)\}$; $\{(D,E,F), (G,H,I)\}$
(e)	0.399	$\{A,B,C,D\}$; $\{D,E,F,G\}$; $\{G,H,I,J\}$; $\{J,L,K,A,G,D\}$
(f)	0.435	$\{K,L,M,G,H,D\}$; $\{A,B,C,D,E,F\}$; $\{D,F,I,J,M,N,O\}$
(g)	0.512	$\{A,B,C,I,J,G,H\}$; $\{C,D,E,K,L,M,N\}$; $\{A,C,E,F,O,P,Q,R\}$
(h)	0.324	$\{P_2,L_2,P_3,L_5,P_6,L_1\}$; $\{P_5,L_3,P_4,L_6,P_6,L_4\}$; $\{P_1,L_1,L_4,P_6\}$
(i)	0.397	$\{L_1,P_4,L_3,P_5,L_4,P_6,L_8,P_{11}\}$; $\{P_9,L_7,P_8,L_6,P_7,L_5\}$; $\{(L_1,L_8,P_6), (P_9,P_8,L_5)\}$; $\{P_1,P_2,P_3,L_0,L_2,L_9,P_{10},P_{13},P_{12}\}$

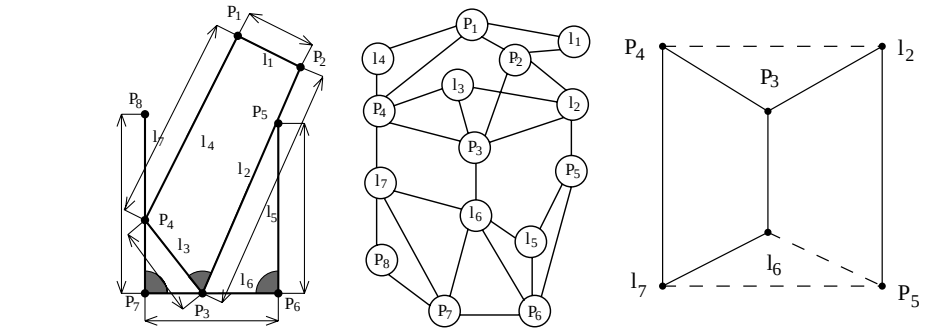

Fig. 16. A 2D constraint problem (a) and its constraint graph.

diagram. In other words, we try to avoid 'jumps.' The problem is solved by comparing the two solutions with the initial positions and then the software will remember the relative position of the relevant elements. For instance, let p be the intersection of two circles with centers o_1 and o_2 in the original figure. After a constraint was changed, we assume that the sign of the signed area of triangle po_1o_2 still keeps the same.

Another concern is the speed. While changing the constraint values, we adopt the following strategies to keep the software fast: (1) after a constraint is changed, we need only to compute the positions for those elements whose positions are affected by the changed constraint in the C-tree. (2) When creating GCs, try to use the constructions which are easier to compute. For instance constructions for points are generally easy to compute than constructions for lines. So we always try to construction lines before points. (3) Optimize the code by using explicit formulas to solve linear and quadratic equations.

Table 1 contains the running time for our software to draw the diagrams and the final GCs in the C-tree. The timings are collected on a PC with a 2.22 GHz CPU.

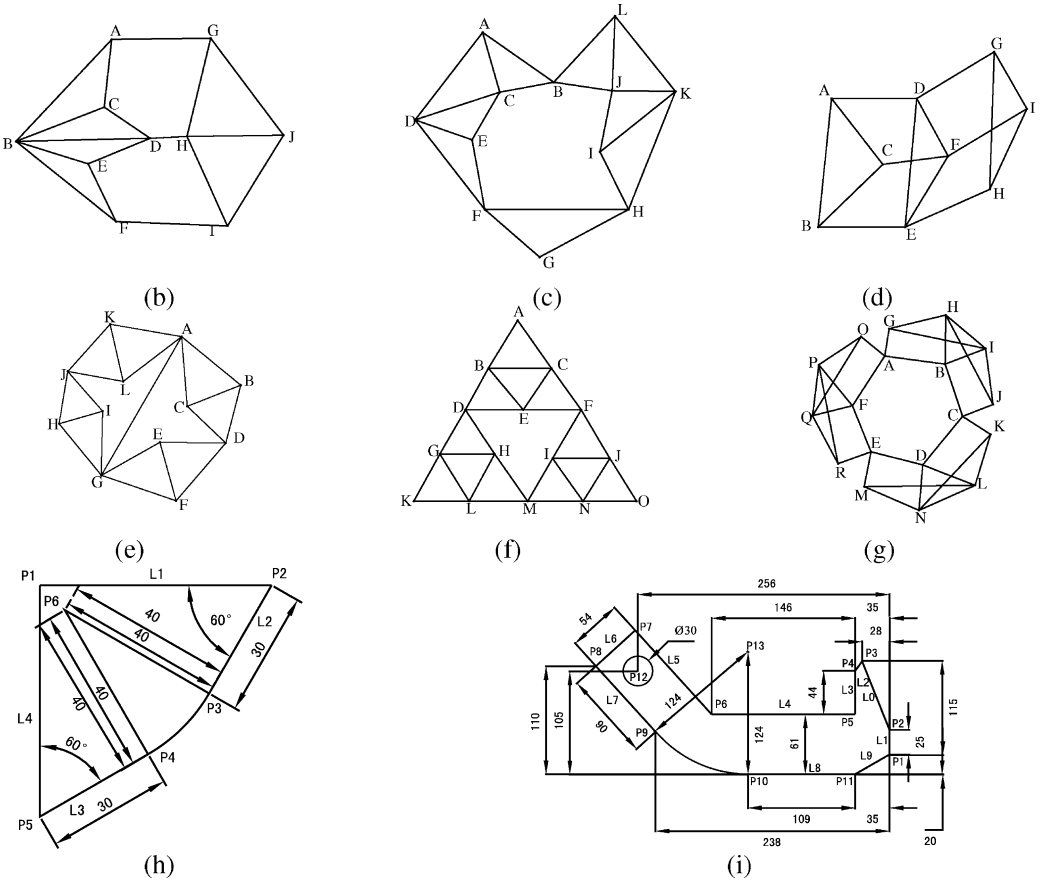

Fig. 17. Eight constraint problems.

We use problem (a) in Fig. 16 to explain the meaning of the data in the table. In Fig. 16, the left figure is the geometric constraint problem and the middle figure is the constraint graph. The final GCs are the leaves of the C-tree. For problem (a), the first GC $\{P_4,P_3,L_3,L_2,P_2,P_1,L_4,L_1\}$ and the second GC $\{P_7,P_6,L_6,L_5,P_5,L_7,P_8\}$ are two explicit construction sequences. The third GC $\{(P_4,P_3,L_2),(L_7,L_6,P_5)\}$ is shown on the right part of Fig. 16, where the dotted lines are the auxiliary constraints. It is a basic merge pattern with $\mathcal{B}=(P_4,P_3,L_2)$ and $\mathcal{U}=(L_7,L_6,P_5)$. In problems (b)–(g) of Fig. 17, the lengths for all the line segments are assumed to be known. The constraint values in problems (h) and (i) are explicitly given.

6. Conclusion

A geometric constraint solving procedure usually consists of two phases: the analysis phase, which is to reduce a large geometric constraint problem into several subproblems, and the computation phase, which is to merge the subproblems by numerical or symbolic computation. In this paper, we propose an analysis method which may be used to decompose any constraint problem into smaller rigid bodies if possible. Comparing to other decomposition methods, our method can be used to handle general constraint problems and is easier to understand and implement. Experimental results show that the algorithm finds the smallest decomposition for all the testing examples efficiently.

The merge phase could be very difficult. This is due to the intrinsic difficulty of the constraint problem: there exist constraint problems of any size which cannot be decomposed into smaller rigid bodies. For these problems, we have to solve them with brutal force computation methods. In our software, we use a numerical computation method based on optimization, which can be used to find solutions of a set of equations consisting of up to 100 quadratic equations in seconds [14].

References

[1] Becker E, Brostein M, Cohen H, Eisenbud D, Gilman R. Algorithms Comput Math; 1999;5:331–51.
[2] Brüderlin B. Using geometric rewriting rules for solving geometric problems symbolically. Theor Comput Sci 1993;116:291–303.
[3] Chou SC. Mechanical geometry theorem proving. Dordrecht, Netherlands: Reidel; 1988.
[4] Dasgupta B, Mruthyunjaya TS. The Stewart platform manipulator. Rev Mech Mach Theor 2000;35:15–40.
[5] Dufourd JF, Mathis P, Schreck P. Geometric construction by assembling solved subfigures. Artif Intell 1998;99:73–119.
[6] Durand C, Hoffmann CM, Systematic A. Framework for solving geometric constraints analytically. J Symbolic Comput 2000;30(5):493–529.
[7] Fudos I, Hoffmann CM. A graph-constructive approach to solving systems of geometric constraints. ACM Trans Graph 1997;16(2):179–216.
[8] Gao XS, Hoffmann CM, Yang WQ. Solving spatial basic geometric constraint configurations with locus intersection. Comput Aided Des 2004;36(2):111–22.
[9] Gao XS, Huang L, Jiang K. A hybrid method for solving geometric constraint problems. In: Richter-Gebert J, Wang D, editors. Automated deduction in geometry. Berlin: Springer; 2001. p. 16–25.
[10] Gao XS, Lei D, Liao Q, Zhang G. Generalized Stewart–Gough platforms and their direct kinematics. IEEE Trans Robot 2005;21(2):141–51.
[11] Gao XS, Zhang G. Classification and solving of merge patterns in geometric constraint solving. Proceedings of shape modeling and applications.: IEEE Press; 2003 p. 89–90.
[12] Gao XS, Zhang G. Geometric constraint solving via C-tree decomposition. ACM SM03, Seattle, USA. New York: ACM Press; 2003 p. 45–55.
[13] Gao XS, Wang D, Qiu Z, Yang H. http://www.mmrc.iss.ac.cn/mmp
[14] Ge J, Chou SC, Gao X. Geometric constraint satisfaction using optimization methods. Comput Aided Des 2000;31(14):867–79.
[15] Hoffmann CM, Chiang CS. Variable-radius circles of cluster merging in geometric constraints. I. Translational clusters. Comput Aided Des 2002;34(11):787–97.
[16] Hoffmann CM, Vermeer PJ. Geometric constraint solving in R^2 and R^3. In: Du DZ, Huang F, editors. Computing in Euclidean geometry. Singapore: World Scientific; 1995. p. 266–98.
[17] Hoffmann CM, Lomonosov A, Sitharam M. Finding solvable subsets of constraint graphs. In: LNCS, No. 1330; 1997, p. 163–97.
[18] Hoffmann CM, Lomonosov A, Sitharam M. Decomposition plans for geometric constraint systems. I. Performance measures for CAD. J Symbolic Comput 2001;31:367–408. Hoffmann CM, Lomonosov A, Sitharam M. Decomposition plans for geometric constraint systems. II. New algorithms. J Symbolic Comput 2001;31:409–27.
[19] Jermann C, Neveu B, Trombettoni G, New A. Structural rigidity for geometric constraint systems. In: Automated deduction in geometry; 2004. p. 87–105.
[20] Joan-Arinyo R, Soto A, Correct A. Rule-based geometric constraint solver. Comput Graph 1997;21(5):599–609.
[21] Joan-Arinyo R, Soto-Riera A, Vila-Marta S, Vilaplana-Pastó J. Revisiting decomposition analysis of geometric constraint graphs. Comput Aided Des 1992;36(2):123–40.
[22] Joan-Arinyo R, Soto-Riera A, Vila-Marta S, Vilaplana-Pastó J. Transforming an under-constrained geometric constraint problem into a well-constrained one. Proceedings of ACM SM03. New York: ACM Press; 2003 p. 33–44.
[23] Kondo K. Algebraic method for manipulation of dimensional relationships in geometric models. Comput Aided Des 1992;24(3):141–7.
[24] Kramer GA. Solving geometric constraints systems: a case study in kinematics. Cambridge, MA: MIT Press; 1992.
[25] Kumar AV, Yu L. Sequential constraint imposition for dimension-driven solid models. Comput Aided Des 2001;33:475–86.
[26] Lamure H, Michelucci D. Solving geometric constraints by homotopy. IEEE Trans Vis Comput Graph 1996;2(1):28–34.
[27] Lamure H, Michelucci D. Qualitative study of geometric constraints, in geometric constraint solving and applications. Berlin: Springer; 1998 p. 234–58.
[28] Latham RS, Middleditch AE. Connectivity analysis: a tool for processing geometric constraints. Comput Aided Des 1996;28:917–28.
[29] Lee JY, Kim K. Geometric reasoning for knowledge-based design using graph representation. Comput Aided Des 1996;28(10):831–41.
[30] Li YT, Hu SM, Sun JG. A constructive approach to solving 3D geometric constraint systems using dependence analysis. Comput Aided Des 2002;34(2):97–108.
[31] Lin VC, Gossard DC, Light RA. Variational geometry in computer-aided design. Comput Graph 1981;15(3):171–7.

[32] Owen JC. Algebraic solution for geometry from dimensional constraints. In: ACM symposium foundation of solid modeling; 1991. p. 397–407.
[33] Owen JC, Power SC. The nonsolvability by radicals of generic 3-connected planar graphs, in automated deduction in geometry. Berlin: Springer; 2004 p. 124–31.
[34] Podgorelec D, New A. Constructive approach to constraint-based geometric design. Comput Aided Des 2002;34(11):769–85.
[35] Verroust A, Schonek F, Roller D. Rule-oriented method for parameterized computer-aided design. Comput Aided Des 1992; 24(10):531–40.

Xiao-Shan Gao is a professor in the Institute of Systems Science, KLMM, Chinese Academy of Sciences. His research interests include: automated reasoning, symbolic computation, intelligent CAD and CAGD, and robotics. He has published over one hundred research papers, two monographs and edited four books or conference proceedings. Webpage: http://www.mmrc.iss.ac.cn/~,xgao

Qiang Lin received his PhD degree from the Chinese Academy of Sciences in 2004. Now, he is an engineer in the Chinese Institute of Electronics Standardization. His research interests are geometric constraint solving, intelligent CAD and computer software.

Gui-Fang Zhang received her PhD degree from the Chinese Academy of Sciences in 2003. Now, she is a postdoctor fellow in the Department of Computer Science and Technology, Tsinghua University. Her research interests are geometric constraint solving and intelligent CAD.