Geometric Constraint Solving via Decomposition and Assembly

Xiao-Shan Gao* and Guifang Zhang†

October 18, 2003

Abstract

In this paper, we propose a general scheme to solve geometric constraint problems. First, we decompose a geometric constraint problem into a c-tree. With this c-tree, solution of the constraint problem is reduced to the solution of general construction sequences. To solve a general construction sequence, we need to solve certain kind of basic merge patterns. One of the basic merge pattern is called generalized Stewart platform, which is to assemble two rigid bodies. We compute the analytical solutions for the 2D general Stewart platforms and a class of 3D general Stewart platforms.

Keywords. Geometric constraint solving, parametric CAD, general construction sequence, decomposition tree, generalized Stewart platform, assembly.

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 parameter values. There are four major approaches to GCS: the numerical approach[12, 25, 30], the symbolic approach[8, 22], the rule-based approach[1, 7, 23, 35] and the graph-based approach [3, 6, 16, 18, 27, 29]. This paper will focus on using graph algorithms to decompose large constraint problems into smaller ones and how to assemble small problems into large ones.

In [33], Owen proposed a GCS method based on the triconnected decomposition of graphs, which may be used to reduce a class of constraint problems into constraint problems consisting of three primitives. In [6, 15], Hoffmann et al proposed a method based on cluster formation to solve 2D and 3D constraint problems. An algorithm is introduced by Joan-Arinyo et al in [19] 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 [27], 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* (definition in Section 2). A similar method based on maximal matching of bipartite graphs was proposed in [26]. In [16], Hoffmann et al gave an algorithm to find rigid bodies in a constraint problem. From this, several general approaches to GCS are proposed [17]. In [14], Jermann et al also gave a general approach to GCS based on the method in [16].

The basic idea for all graph based methods is to reduce a large constraint problem into several smaller ones. Among these smaller problems, the largest (with largest degrees of freedom) is called the *controlling problem*. The controlling problem can be used to measure the effect of the decomposition method.

In this paper, we propose a general scheme to solve geometric constraint problems, which consists of three steps.

First, we propose an algorithm to decompose a geometric constraint problem into a c-tree. With this c-tree, solution of the constraint problem is reduced to solution of general construction sequences. The main reason for introducing the c-tree is that we may obtain smaller controlling problems than using general construction sequence alone. We may obtain the c-tree with the smallest controlling problem for a constraint problem if we do an exhaust search. But this will increase the average complexity of the method by a factor of $O(e)$ where e is the number of constraints.

Second, we need to solve a general construction sequence, which can be reduced to to solve certain kind of basic merge patterns. One of the basic merge pattern is called generalized Stewart platform, which is to assemble two rigid bodies. We compute the analytical solutions for the 2D general Stewart platforms and a class of 3D general Stewart platforms.

At last, two solved rigid bodies are assembled together to obtain the solution to the original problem.

Parts of the results in this paper have been reported at SM03 [10] and SMI03 [11].

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 [14, 17, 23, 26, 27],

*Institute of Systems Science, Academia Sinica, xgao@mmrc.iss.ac.cn
†Department of Computer Science, Tsing-Hua University, gfzhang@mmrc.iss.ac.cn

the method MFA proposed in [17] and the c-tree method can be used to find a decomposition with the smallest controlling problem in certain sense. The MFA and c-tree methods have the same complexity. Both of them can be used to solve 2D and 3D problems, although paper [17] focuses on the 2D case and this paper focuses on the 3D case.

2 General Construction Sequence

2.1 Basic Concepts about Constraint Graphs

We consider two types of *geometric primitives*: points and lines in two dimensional Euclidean plane and two types of constraints: the distance constraints between point/point, point/line and angular constraint between line/line; three types of *geometric primitives*: points, planes and lines in the three dimensional Euclidean space and two types of 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 $\text{ANG}(o_1, o_2)$ and $\text{DIS}(o_1, o_2)$ respectively. We will use p_i, h_i and l_i to represent points, planes and lines respectively.

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. Figure2 is the graph representation for the constraint problem in Figure 1.

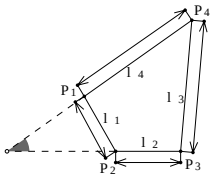

Figure 1: Lengths of four edges and angle (l_2, l_4) are given

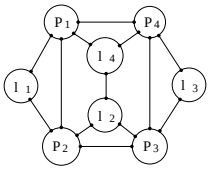

Figure 2: Graph representation for the problem

For an edge e in a constraint graph, let $\text{DOC}(e)$ 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 one. There are several exceptions: (1) Constraint $\text{DIS}(p_1, p_2) = 0$. In this case, $p_1 = p_2$. In 2D the constraints has valence two; in 3D the constraint has valence three. We assume that this case does no occur. (2) Constraint $\text{DIS}(p_1, l_1) = 0$ has valence two in 3D. (3) Constraint $\text{ANG}(h_1, h_2) = 0$ has valence two in 3D. (4) Constraint $\text{ANG}(l_1, l_2) = 0$ has valence two in 3D. (5)$\text{DIS}(l_1, l_2) = 0$ has valence two in 2D. These constraints are *degenerate cases*.

For a geometric primitive o, let $\text{DOF}(o)$ be the degrees of freedom for o, which is the number of independent parameters required to determine the geometric primitive. For a constraint graph G, let $\text{DOF}(G) = \sum_{v \in \mathbf{V}(G)} \text{DOF}(v)$, $\text{DOC}(G) = \sum_{e \in \mathbf{E}(G)} \text{DOC}(e)$. In the constraint graphs, we use n lines to represent a constraint of valency n.

Generally, we call a constraint system *geometrically well-constrained* if the shape of the corresponding diagram has only a finite number of cases; *geometrically under-constrained* if the shape of the corresponding diagram has infinite solutions; *geometrically over-constrained* if the corresponding diagram has no solution.

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

In most cases a structurally well-constrained graph is geometrically well-constrained and hence defines a rigid body. But, in some special cases the constraint problem represented by a structurally well-constrained graph may have no solutions or an infinite number of solutions. For detailed studies on the relation between the two kinds of constraint problems, please consults [13, 34]. In this paper, we will concern the structure solvability of the constraint problem only. Therefore, when we say rigid bodies, we mean structurally well-constrained problems.

2.2 General Construction Sequence

A *general construction sequence* (GC) for a constraint problem $\mathcal{G}$ is a sequence:

$$C_1, C_2, \cdots, C_n$$

where each C_i is a set of geometric primitives in $\mathcal{G}$, such that

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

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

The maximal of $\text{DOF}(C_i)$ for $i = 1, \cdots, n$ 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 the general construction sequence $\mathcal{C}$ and denoted by MDOF($\mathcal{C}$).

In [27], Latham and Middleditch proposed an algorithm which may be used to decompose a well-constrained problem into a general construction sequence. In [17], Hoffmann et al proposed an algorithm of complexity $O(n(e+n))$ which can also be used to decompose a well-constrained problem into a general construction sequence, where n is the number of vertices and e is the number of edges in the constraint graph.

Before using these algorithms, we need to add three more degrees of freedom to a set of primitives in 2D and six more degrees of freedom to a set of primitives in 3D. These primitives are called *base primitives*. The geometric meaning of this step is as follows: a rigid body in the plane has three degree of freedom and a rigid body in the space has six degrees of freedom. By selecting the base primitives, we can find the absolute position of the rigid body in the plane or the space. After this step, a structurally well-constrained problem G will satisfy DOC(G) = DOF(G), which is called *strictly well-constrained*. If the modified constraint graph is not strictly well-constrained, the connectivity algorithm of Latham and Middleditch may be used to add or delete a proper number of constraints to obtain a strictly well-constrained problem. In [20], an algorithm was given to obtain a well-constrained problem from an under-constrained problem if the constrain problem can be decomposed into an s-tree.

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 distance constraint as the base primitives. For instance, 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.

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

$\mathcal{G}_1$: $\{P_1\}, \{P_4\}, \{l_4\}, \{P_1, P_2, P_3\}, \{l_1\}, \{l_3\}$

$\mathcal{G}_2$: $\{P_1\}, \{P_2\}, \{P_4, P_3, l_2, l_4\}, \{l_1\}, \{l_3\}$

If using GCs $\mathcal{G}_1$ and $\mathcal{G}_2$ to solve the problem, we have MDOF($\mathcal{G}_1$) = DOF($\{p_1, p_2, p_3\}$) = 6, MDOF($\mathcal{G}_2$) = DOF($\{p_4, p_3, l_2, l_4\}$) = 8. It is clear that $\mathcal{G}_1$ is better.

2.3 Find Base Primitives in 3D

To find base primitives for a 3D constraint problem, we first try to find a rigid body consisting of less than four primitives in the constraint problem. The four graphs in Figure 3 represent such rigid bodies in 3D.

If such a rigid body does not exist, we try to find three

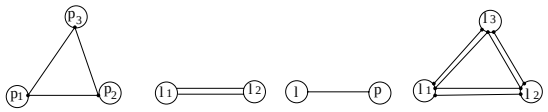

Figure 3: Rigid bodies with two or three primitives in 3D

points p_1, p_2, p_3 such that $d_1 = \text{DIS}(p_1, p_2)$ and DIS(p_2, p_3) are known. We select p_1, p_2, p_3 as the base primitives by adding the following constraints: $p_1 = (0, 0, 0)$, $p_2 = (d_1, 0, 0)$, $p_3 = (x, y, 0)$, where x and y are variables to be determined. Other cases can be treated similarly.

In what below, let us show how to find the first diagram in Figure 3.

1. We search all the edges $e = (o_1, o_2)$ representing a distance constraint in the constraint graph.
2. For each e, we search all the vertices o having a distance constraint with o_1.
3. If o_2 has a distance constraint with o, then o_1, o_2, o form a rigid body and can be treated as base primitives. The algorithm terminates.

Let e be the number of edges $l_1, \cdots, l_e$ 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

$$\sum_{i=1}^{e} \sum_{j=1}^{d_i} O(1) = \sum_{i=1}^{e} O(d_i) = O(\sum_{i=1}^{e} d_i) = O(e).$$

The last step is true because $\sum_{i=1}^{e} d_i = 2e$, since each constraint involving two primitives. All other cases can be treated similarly.

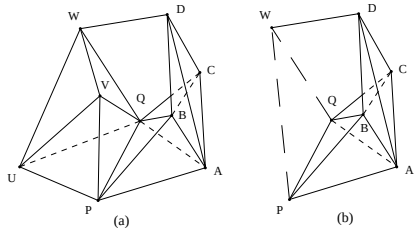

Figure 4: A 3D constraint problem

Let us look at the constraint problem in Figure 4(a), where each line represents a distance constraint between two points. We need only to determine the position of the points.

For this problem, there are three essentially different GCs $\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_3$, the geometric meaning of which is self evident.

$\mathcal{C}_1$: $\{P\},\{Q\},\{A\},\{B\},\{C\},\{D\},\{U,V,W\}$

$\mathcal{C}_2$: $\{P\},\{Q\},\{U\},\{V\},\{W\},\{A,B,C,D\}$

$\mathcal{C}_3$: $\{W\},\{D\},\{P,Q,A,B,C,D,U,V\}$

It is clear that the generated GCs depends on the base primitives. The base primitives for GCs $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ are $\{P,Q,A\}$, $\{P,Q,U\}$, $\{W,D,C\}$. Actually, according to our methods of selecting base primitives, only $\mathcal{C}_1$ and $\mathcal{C}_2$ could be generated. $\mathcal{C}_3$ will not be generated.

If using GCs $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ to solve the problem, we have MDOF($\mathcal{C}_1$) = DOF($\{U,V,W\}$) = 9, MDOF($\mathcal{C}_2$) = DOF($\{A,B,C,D\}$) = 12, MDOF($\mathcal{C}_3$) =, DOF($\{A,B,C,D,U,V,W\}$) = 21. It is clear that $\mathcal{C}_1$ is the best GC.

3 A C-tree Decomposition

3.1 A New Decomposition Tree: C-tree

The concept of s-tree is introduced by Joan-Arinyo et al in [19] to decompose a 2D constraint problem into triangles and tri-connected components. The s-tree is equivalent to the decomposition tree of Owen [33] and Fudos and Hoffmann's cluster merging method [6], but is conceptually simpler. Basically speaking, the s-tree handles problems that can be decomposed into the form of triangles. We will introduce a new decomposition tree, c-tree, that can be used to simplify general constraint problems.

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. $\mathbf{V}(S) = (\mathbf{V}(G) - \mathbf{V}(H)) \cup I$. All the edges in $\mathbf{E}(G)$ between two vertices in $\mathbf{V}(S)$ will be in $\mathbf{E}(S)$. If S is structurally well-constrained, S is the split subgraph. Otherwise, we add DOF($\mathbf{V}(S)$) − DOC($\mathbf{E}(S)$) − 6 *auxiliary constraints* between vertices in I to make the new graph S structurally well-constrained. This can be done with the algorithm in [27]. This new graph S is called the split subgraph.

For example, let G be the graph in Figure 4(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 Figure 4(b), where the two constraints between W/P and W/Q are the auxiliary constraints. 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 two auxiliary constraints between W/P and W/Q to make S a well-constraint problem. Then the solution of G is reduced to the solution of two smaller problems H and S.

Definition 3.1 *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 either a faithful subgraph of N and R is the split subgraph of N with L.*
2. *L is a GC for N and $R = \emptyset$.*

All leaves are GCs.

Continue with the example in Figure 4(a). $H = \{W,U,V,P,Q\}$ is a faithful subgraph. The split subgraph S is the one in Figure 4(b). Then H and S are the left and right children of G in the c-tree in Figure 5. The left children for H and S are their GCs respectively.

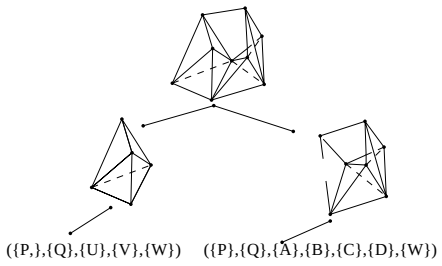

Figure 5: A c-tree for the problem in Figure 4

We may say that the c-tree is a natural generalization for the s-tree from [19]. 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. We do a left to right depth-first search of the c-tree and consider the following three cases.

1. The current node N is a GC. We need to compute the basic merge patterns in this GC to solve the constraint problem represented by the farther node of N.
2. The current node N only has the left child L, which is a GC. In this case, L is evaluated and N is solved.
3. The current node has two children. In this case, we already solved the left child which is a rigid body. From this rigid body, we may compute the numerical values for the auxiliary constraints in the right child. Now the right child becomes a structurally well-constrained problem. We may solve the right child recursively.

4. The merging of the left and right children is easy, since they are connected by sharing several geometric primitives.

It is clear that the computation of a c-tree is reduced to the computation of GCs and hence to basic merge patterns.

To solve the problem in Figure 4(a) with the c-tree in Figure 5, we first compute the left child of the root using the construction sequence $\{P\}, \{Q\}, \{U\}, \{V\}, \{W\}$. Then, we may compute the distances between P/W and Q/W and solve the right child of the root similarly.

3.2 An Algorithm to Find a C-tree

Algorithm 3.2 *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 for T and to generate a new graph H by adding more constraints as follows. If the base primitives form a rigid body as shown in Figure 3, we may add six new constraints to fix the position of this rigid body. For instance, if the primitives form a triangle $P_1P_2P_3$ the length of whose three edges are known, we let $P_1 = (0, 0, 0)$, $P_2 = (x_2, 0, 0)$, $P_3 = (x_3, y_3, 0)$. In other words, we add three position constraints to P_1, two direction constraints to P_2 and one direction constraint to P_3. Other cases can be treated similarly.

S2 With Latham-Middleditch's[27] or Hoffmann's[17] algorithm , we may find a GC for H

$$\mathcal{C}: \quad C_1, \cdots, C_m.$$

Using Proposition 5.2 to decide whether $\mathcal{C}$ is angular conflict. If it is, the problem has angular conflicts and the algorithm terminates.

S3 Let $\mathcal{B}_i = \cup_{j=1}^{i} C_j$. If $\mathrm{CN}(\mathcal{B}_i, C_{i+1}) < 5$, C_{i+1} is a point, a plane or a line. These cases are relatively easy to solve. Merge C_i and C_{i+1} into one set. After all such merges, we obtained a *reduced GC*:

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

S4 Let $\mathcal{B}'_i = \cup_{j=1}^{i} C'_j$. 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 k be the smallest number satisfying the following conditions. There exists at least one primitive $o \in \mathcal{B}'_k$ such that there are no constraints between o and primitives in $C'_i, i = k+1, \cdots, s$. 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 goto 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 $C_1, \cdots, C_d$ where d is an integer satisfying $\mathcal{B}_d = \mathcal{B}'_s$; the right child of F is the empty set. The right child of T is the split subgraph G' of G with F. Set $T = G'$ goto S1.

Let n and e be the number of vertices and edges in G. As mentioned in Section 2.2, Step S1 has complexity $O(e)$. In S2, Latham-Middleditch's algorithm uses the maximal b-matching from graph theory, which has complexity $O(n(n+e))$ [17]. Steps S3, S4 and S5 are also 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 primitive sets in Figure 3 is linear in terms of e. The loop started at S7 could run n times. So the total complexity of the algorithm is $O(n^2(n+e)e)$. Please be noted that this is also the complexity under which we may find a decomposition with the smallest controlling sub-problem. In most cases, the loop started at S6 will only run for one or two times.

Since the computation of a c-tree T is reduced to the computation of GCs, we define $\mathrm{MDOF}(T)$ to be maximal $\mathrm{MDOF}(C)$ for all C which are the GCs in T. A c-tree T for constraint graph G is called *minimal* if $\mathrm{MDOF}(T)$ is the smallest for all possible c-trees for G. We may modify Algorithm 3.2 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 the graph decomposition is unique [27]. Therefore, by searching all the possible base primitives, we have obtained all the GCs and thus all the possible c-trees for the problem. The complexity for this modified algorithm is still $O(n^2(n+e)e)$, because in the worst case, to find a faithful subgraph we also need to search all the base primitives. But, in most cases, to find a faithful subgraph, we need only to search one or two sets of base primitives. Therefore, in practice, we generally satisfies to find a c-tree. Roughly, speaking, this will reduce the running steps by a factor of e.

3.3 Working Examples

Let G be the graph in Figure 4(a), which is also the root of the c-tree. In Step S1 of Algorithm 3.2, we select P, Q, U as the base primitives. In other words, we will construct G starting from these points. In S2, a GC

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

for H is generated. In S3, the single points in $\mathcal{C}$ is collected together to form the followiong reduced GC.

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

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 $C_1' = \{P,Q,U,V,W\}$ is a faithful subgraph. In S7, new notes are added to the c-tree. The left child of the root is C_1' and the left child of C_1' is the following construction sequence:

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

The right child of the root is the split subgraph of H by C_1', which is the graph in Figure 4(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:

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

Now the c-tree in Figure 5 is generated. Basically speaking, to solve the problem, we need to solve two GCs: $\mathcal{C}_4$ and $\mathcal{C}_5$. Since MDOF(C_4) = MDOF(C_5) = 3, which is the simplest case we could have. This solution is the same as that obtained with the cluster formation algorithm of Hoffmann et al [15].

Figure 6(a) is a more difficult constraint problem, where each edge represents a distance between two points. Figure 6(b) is the c-tree for it. In Algorithm 3.2, we select points A, B, C as the base primitives. Latham-Middleditch's algorithm will give a GC as follows:

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

The reduced GC in S3 is

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

In S5, $k = 2$, which will lead to the children of the root (Figure 6(b)).

Note that the problem in Figure 4(a) could be solved with the cluster merging method proposed in [15]. Another possible way to solve the problem in Figure 4(a) is to decompose the constraint graph into 4-connected components with the algorithm from [21]. But the problem in Figure 6 cannot be simplified with both methods. For the cluster merging method, there exist no clusters. For the 4-connected method, this graph cannot be divided into two 4-connected parts. Actually, all the problems that could be solved with the cluster merging method in [15] can also be solved similarly with our method. A strict proof would be too long to be included in this paper. In what below, we will give a sketch of the proof. A cluster is a rigid body that can be solved with a construction sequence. When we choose three primitives in a cluster, the GC generated will include the construction sequence for the cluster as a sub-sequence. This sub-sequence will form a faithful subgraph and will lead to a division of the problem into two sub-problems.

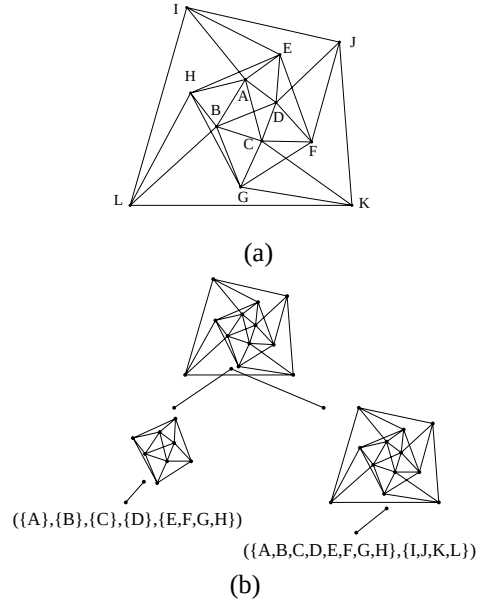

Figure 6: A constraint problem about twelve points

4 Solving of General Construction Sequences in 2D

With a c-tree decomposition, solving of a constraint problem is reduced into the solving of a GC. In this section, we will show how to solve a GC. Suppose that a constraint graph G is decomposed into a GC:

$$\mathcal{C}: \quad C_1, C_2, \cdots, C_n \tag{1}$$

We call the type of dependency of C_i on $C_1, \cdots, C_{i-1}$ a *basic merge pattern*. Let

$$\mathcal{B} = \bigcup_{k=1}^{i} C_k, \qquad \mathcal{U} = C_{i+1}.$$

We call $\mathcal{B}$ and $\mathcal{U}$ *the base objects* and *the dependent objects*, respectively. To solve a GC, we need to determine $\mathcal{U}$ assuming that $\mathcal{B}$ is known. The sum of DOC(e) for all edges e between $\mathcal{B}$ and $\mathcal{U}$ describes an important natural of the merging step, and is called *the connection number*, denoted by CN($\mathcal{B}$,$\mathcal{U}$).

Let graph $\mathcal{U} = (V, E)$, where V is the set of vertices and E is the set of edges.

4.1 Classification of GC in 2D

Theorem 4.1 *We have:* $2 \leq CN(\mathcal{B},\mathcal{U}) \leq |V|$.

Proof: Since V contains at least one element and $\mathcal{B}$ is a rigid body, $|CN(\mathcal{B},\mathcal{U})| \geq 2$. The definition of GC guarantees that $\mathcal{U}$ satisfies the following three conditions [27].

1. A vertex v in V represents a point or a line. Hence DOF$(v) = 2$;
2. If $|V| > 1$ then for every vertex v in V there exists at least one constraint between v and a primitive in V. Otherwise, v can be determined by $\mathcal{B}$ alone, which contradicts to the minimum property of V (the second condition on the definition of GC).
3. $\mathcal{U}$ is strongly connected.

Due to the second condition above, there exist at least $2 \times \frac{|V|}{2} = |V|$ constraints between primitives in E, i.e. DEG$(V) \geq |V|$. Since both $\mathcal{B}$ and $\mathcal{B} \cup \mathcal{U}$ are rigid bodies, we need exactly DOF$(V) = \Sigma_{v \in V} DOF(v) = 2|V|$. constraints to determine $\mathcal{U}$. In other words, we have CN$(\mathcal{B},\mathcal{U})+$ DEG$(V) = 2|V|$. Thus CN$(\mathcal{B},\mathcal{U}) = 2|V| -$ DEG$(V) \leq |V|$.

The computation of $\mathcal{U}$ with respect to $\mathcal{B}$ can be divided into the following three cases.

1. If CN$(\mathcal{B},\mathcal{U}) = 2$, $\mathcal{U}$ consists of one geometric element o, and o can be constructed explicitly.
2. If CN$(\mathcal{B},\mathcal{U}) = 3$, $\mathcal{U}$ has $2|V| - 3$ constraints between primitives in V. Hence $\mathcal{U}$ is a rigid body. It is to assembly two rigid bodies according to three constraints. We give complete analytical solution to this case in Section 4.2.
3. If 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 $\mathcal{U}$. As an example, the basic merging patterns for the case $|\mathcal{U}| = 5$ is shown in Figure 7.

In what below, 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 $\mathcal{U}$.

4.2 Analytic Solutions to Generalized Stewart Platform in 2D

When CN$(\mathcal{B},\mathcal{U}) = 3$, both $\mathcal{B}$ and $\mathcal{U}$ in the basic merging pattern are rigid bodies. We call basic merging pattern *generalized Stewart platform*(GSP), $\mathcal{B}$ the base and $\mathcal{U}$ the platform.

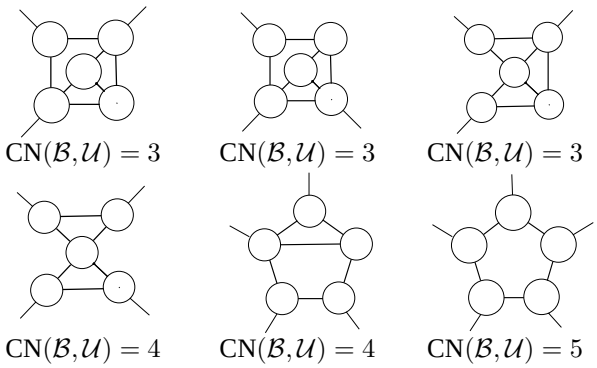

Figure 7: Basic merge patterns of five primitives

Figure 8 is the illustration of a 2D GSP. The problem can be divided into two cases: the **ddd** case in which all the three constraints between $\mathcal{B}$ and $\mathcal{U}$ are distance constraints; the **dda** case in which two of the constraints are distance constraints and one constraint is an angle constraint. We cannot have more than one angle constraint due to the fact that a 2D rigid body only need one angle constraint to determine its direction.

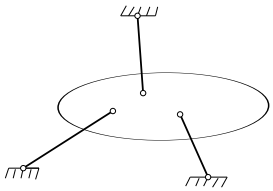

Figure 8: the Stewart Platform in 2D

4.2.1 The Case of dda

We make use of the fact that rotation constraints can violate previous imposed distance constraints while translation doesn't violate previous imposed angular constraints[24]. We impose the angular constraint first to remove the rotation degree of freedom. This will give us a pure geometric solution based on ruler and compass construction.

Imposing Angular Constraint

Let line l_1 be on the base $\mathcal{B}$ and line l_2 on the platform $\mathcal{U}$, where $\angle(l_1, l_2) = \alpha$. Let the angular constraint be $\angle(l_1, l_2) = \beta$. l_2 should rotate angle $\theta = \beta - \alpha$ around a point on l_2. Thus we can get the following rotation matrix

$$\mathbf{R} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}.$$

Imposing Distance Constraints

We will make use of the following definition and theorem in Kumar et al[24]. This theorem is also proposed by us in [7], under the name of 'translational' transformation.

Definition 4.2 *The translation space of a point on the dependent object with respect to a distance constraint is the set of points to which the point can be moved to by translating the dependent object without violating the constraint.*

Theorem 4.3 *If $\mathbf{R}^X(u)$ is the translation space of p_0 on the dependent object with respect to a constraint X, the translation space of any other point p on the dependent object with respect to this constraint is $\mathbf{R_p}^X(u) = \mathbf{R}^X(u) + (p - p_0)$, assuming that previously imposed angular constraints have eliminated all the rotation degree of freedom of the dependent .*

We may now give the following procedure for solving a **dda** problem.

Algorithm 4.4 *The input is a* **dda** *constraint problem with $\mathcal{B}$ and $\mathcal{U}$ as the base and the platform. We will give a geometric construction procedure based on ruler and compass to solve the problem.*

S1 We first fix the rotational degree of freedom of $\mathcal{U}$ by imposing the angular constraint.

S2 We will determine the translation spaces for the objects on the platform according to the distance constraints left. This is divided into three cases.

1. If the distance constraint $|p_0p| = r_0$ is between a point p_0 on the base and a point p on the platform, then the translation space is the following circle
$$R^C(\theta) = p_0 + r_0(\cos\theta\mathbf{i} + \sin\theta\mathbf{j}).$$
2. The distance constraint $|l_0p| = r_0$ is between a line l_0 on the base and a point p on the platform. The translation space is a line if $r_0 = 0$. Otherwise it's two paralleling lines.
$$R^l(u) = \mathbf{p_0} + u\mathbf{I} \pm r_0\mathbf{m}$$
where p_0 is a point on l_0, $\mathbf{I}$ is a unit vector parallel to l_0 and $\mathbf{m}$ is a unit vector perpendicular to $\mathbf{I}$.
3. The distance constraint $|p_0l| = r_0$ is between a point p_0 on the base and a line l on the platform. In this case, we take an arbitrary point p in l and will determine the translation space for p. The constraint $|p_0l| = r_0$ implies that p is a tangent line of the circle with p_0 as center and as r_0 as the radius. Also, we know the direction of l since the rotational degree of freedom of the platform is fixed. Then the position l^* for l can be easily determined by the two constraints. Since l is a free point on line l, l^* is the translation space for l.

S3 Since $\mathcal{U}$ has a fixed direction, we need only to compute the corresponding coordinate p^* of a point p on $\mathcal{U}$. Then we need only to translate $\mathcal{U}$ by $p^* - p$ to move it to the correct position. By S2, for each distance constraint $d_i, i = 1, 2$, we have a point p_i and a translation space $\mathbf{R}^{d_i}$ for p_i. By Theorem 4.3, the position for p_2 is the intersection of the following two translation spaces, which are either lines or circles.
$$\begin{cases} \mathbf{R}_{p_2}^{d_1}(u_1) = \mathbf{R}^{d_1}(u_1) + (p_2 - p_1) \\ \mathbf{R}_{p_2}^{d_2}(u_2) = \mathbf{R}^{d_2}(u_2). \end{cases} \quad (2)$$

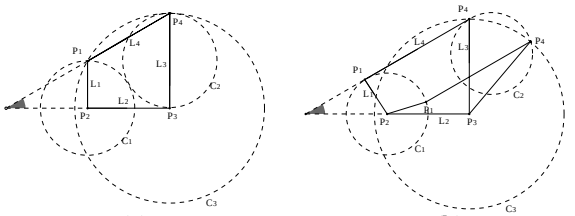

Figure 9: Geometric solution to the problem in Figure 1.

For the example in Figure1, l_1 and l_2 can be constructed easily if the remaining objects are solved. After we delete l_1 and l_3 from the graph in Figure 9, the graph consists of two rigids: $p_1p_4l_4$ and $p_2p_3l_2$. Let $p_1p_4l_4$ be the platform and $p_2p_3l_2$ the base respectively. We construct the translation space of distance constraint d_1 between point p_1 and p_2, which is circle c_1 shown in Figure 9. Then we can construct the translation space for distance constraint d_2 between point p_3 and p_4, which is circle c_3 shown in Figure 9. According to Theorem 4.3 we can get the translation space of point p_4 with respect constraint d_1, which is circle c_2. The intersection of circle c_2 and circle c_3 is the solution to point p_4^*. We can translate rigid $p_1p_4l_4$ along the vector $\mathbf{t} = p_4^* - p_4$ and position rigid $p_1p_4l_4$. The problem could have one or two solutions shown in Figure 9.

In Step S3, we need to find the intersection of lines/circles and lines/circle. The details of the computation is omitted.

4.2.2 The Case of ddd

This case can be divided into 10 different sub-cases shown in Table 3.

Constrain type	Vertices in platform	Vertices in base
PPP-LLL	three points	three lines
PPP-LLP	three points	two lines, one point
PPP-LPP	three points	one lines, two points
LLL-PPP	three lines	three points
LLP-PPP	two lines, one point	three points
LPP-PPP	one line, two points	three points
LPP-PLL	one line, two points	one point, two lines
LLP-PPL	two lines, one point	two points, one line
LPP-PLP	one line, two points	two points, one line
PPP-PPP	three points	three points

Table 3 Ten Cases of the **ddd** Problem

We can always get three points in the base and three points in the platform, respectively. If the involved object is a line, we can take a point on it. Letp_{B_1}, p_{B_2} and p_{B_3} be three points on the base, and p_{D_1}, p_{D_2} and p_{D_3} three points on the platform. p_{B_1}, p_{B_2} and p_{B_3} are not collinear, and p_{D_1}, p_{D_2} and p_{D_3} are not collinear, either.

Let p_{B_1} be the origin of the fixed coordinate system in the base , $p_{B_1}p_{B_2}$ the x-axis. The coordinates of three points in the base are $p_{B_1} = (0,0), p_{B_2} = (b_1,0)$ and $p_{B_3} = (b_2,b_3)$.

Let the point p_D be the origin of the moving coordinate system in the platform. The coordinate of point p_D in the fixed coordinate system is $p_D = (x_3,x_4)$, and point p_D is the foot of perpendicular of point p_{D_3} to $p_{D_1}p_{D_2}$. Let $\angle(p_{B_1}p_{B_2}, p_{D_1}p_{D_2}) = \theta$, $x_1 = \cos\theta, x_2 = \sin\theta$. The moving coordinates of three points in the platform are $p_{D_1} = (-h_1,0), p_{D_2} = (h_2,0), p_{D_3} = (0,h_3)$, where h_1,h_2,h_3 are three nonnegative parameters. $p_{D_1}p_{D_2}$ is the x-axis of the moving coordinate system. Their coordinates in the fixed coordinate system are

$$\begin{cases} p_{D_{11}} = (-h_1x_1 + x_3, -h_1x_2 + x_4) \\ p_{D_{22}} = (h_2x_1 + x_3, h_2x_2 + x_4) \\ p_{D_{33}} = (-h_3x_2 + x_3, h_3x_1 + x_4) \end{cases}$$

There exist at most three lines in the base which satisfy the three distance constraints. Let the parametric equations of these lines be

$$\begin{cases} l_1 : p = p_{B_3} + u_1\mathbf{s_1}, (\mathbf{s_1} = (l_1,m_1), |\mathbf{s_1}| = 1 \\ l_2 : p = p_{B_2} + u_2\mathbf{s_2}, (\mathbf{s_2} = (l_2,m_2), |\mathbf{s_2}| = 1 \\ l_3 : p = p_{B_1} + u_3\mathbf{s_3}, (\mathbf{s_3} = (l_3,m_3), |\mathbf{s_3}| = 1) \end{cases}$$

Because a line can be completely determined by a point on the line and its direction, assuming p_{B_1} is the intersection point of lines l_1 and l_2, we take point p_{B_1} as the point on line l_1. Thus the parametric equation of line l_1 becomes

$$l_1 : p = p_{B_1} + u_1\mathbf{s_1}.$$

Then the number of the parameters is decreased and the equations are simplified. We treat all the following parametric equations of lines alike to simplify the equations by the same way.

There exist at most three lines in the platform which satisfy the three distance constraints. Let the parametric equations of these lines be

$$\begin{cases} l_{01} : p = p_{D_3} + u_1\mathbf{s_1}, (\mathbf{s_1} = (l_1,m_1), |\mathbf{s_1}| = 1 \\ l_{02} : p = p_{D_2} + u_2\mathbf{s_2}, (\mathbf{s_2} = (l_2,m_2), |\mathbf{s_2}| = 1 \\ l_{03} : p = p_{D_1} + u_3\mathbf{s_3}, (\mathbf{s_3} = (l_3,m_3), |\mathbf{s_3}| = 1) \end{cases}$$

Although we use the same $\mathbf{s_i}$ in $_i$ and l_{0i} $(i = 1,2,3)$, there will be no confusion, because the line l_i and l_{0i} $(i = 1,2,3)$ will not appear in the same cases of **ddd** simultaneity. Let l_{01}, l_{02}, l_{03} be three lines in the platform, their parametric equations are

$$\begin{cases} l_{11} : p = p_{D_{33}} + u_1\mathbf{s_{11}}, |\mathbf{s_{11}}| = 1, \\ \qquad \mathbf{s_{11}} = (l_1x_1 - m_2x_2, l_1x_2 + m_1x_1) \\ l_{22} : p = p_{D_{22}} + u_2\mathbf{s_{22}}, |\mathbf{s_{22}}| = 1, \\ \qquad \mathbf{s_{22}} = (l_2x_1 - m_2x_2, l_2x_2 + m_2x_1) \\ l_{33} : p = p_{D_{11}} + u_3\mathbf{s_{33}}, |\mathbf{s_{33}}| = 1, \\ \qquad \mathbf{s_{33}} = (l_3x_1 - m_3x_2, l_3x_2 + m_3x_1) \end{cases}$$

We use $|\mathbf{pl}| = d_{ij}$ $(i = 1,\cdots 9; j = 1,2,3)$ to denote the distance between point p and line l, and $|pp| = t_{ik}$ $(i = 1,\cdots 9; k = 1,2)$ to denote the distance between two points. It is obvious that d_{ij} and t_{ik} should be nonnegative parameters.

We use Wu-Ritt's zero decomposition method [36] to find the analytical solutions. This method may be used to represent the zero set of a polynomial equation system as the union of zero sets of equations in *triangular form*, that is, equation systems like

$$f_1(u,x_1) = 0, f_2(u,x_1,x_2) = 0, \ldots, f_p(u,x_1,\ldots,x_p) = 0$$

where the u could be considered as a set of parameters and the x are the variables to be determined. As shown in [36], solutions to an equation system in triangular form are well-determined. For instance, the number of solutions to an equation system in triangular form can be easily computed.

Case PPP-LLL If the constraints are $|p_{D_{11}}l_3| = d_{13}$, $|p_{D_{22}}l_2| = d_{12}$, and $|p_{D_{33}}l_1| = d_{11}$, we have the following equations:

$$\begin{cases} x_1^2 + x_2^2 - 1 = 0 \\ |m_3(-h_1x_1 + x_3) - l_3(-h_1x_2 + x_4)| - d_{13} = 0 \\ |m_2(h_2x_1 + x_3 - b_1) - l_2(h_2x_2 + x_4)| - d_{12} = 0 \\ |m_1(-h_3x_2 + x_3 - b_2) - l_1(h_3x_1 + x_4 - b_3)| - d_{11} = 0 \end{cases} \tag{3}$$

We may simplify the equation system as follows.

$$\begin{cases} x_1^2 + x_2^2 - 1 = 0 \\ s_4x_1 + m_3x_3 + s_5x_2 - l_3x_4 + s_1 = 0 \\ s_6x_1 + m_2x_3 - s_7x_2 - l_2x_4 + s_2 = 0 \\ s_8x_2 + m_1x_3 - s_9x_1 - l_1x_4 + s_3 = 0 \end{cases} \tag{4}$$

where $s_1 = \pm d_{13}, s_2 = -m_2b_1 \pm d_{12}, s_3 = -m_1b_2 + l_1b_3 \pm d_{11}, s_4 = -m_3h_1, s_5 = l_3h_1, s_6 = m_2h_2, s_7 = l_2h_2, s_8 = -m_1h_3, s_9 = l_1h_3$.

We may simplify the equation system as follows.

$$\begin{cases} z_{111}x_4 + z_{112}x_1 + z_{113} = 0 \\ z_{121}x_3 + z_{122}x_1 + z_{123} = 0 \\ z_{131}x_2 + z_{132}x_1 + z_{133} = 0 \\ z_{141}x_1^2 + z_{142}x_1 + z_{143} = 0 \end{cases} \tag{5}$$

where the z_{ijk} are polynomials free of x_1, $\cdots$, x_4. The detailed expressions for coefficients z_{ijk} may

be found in Appendix A, which may be found in http://www.mmrc.iss.ac.cn/˜ xgao/publ.html.

The above analytical solutions reduce the problem to the solving of one quadratic equation and three linear equations. There are at most 16 solutions, since s_1, s_2, s_3 could take two opposite values and the forth equation is of degree 2.

We omit the other nine cases which can be found in [11].

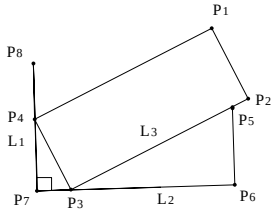

Figure 10: An example of case ddd

Example 4.5 *With the method motioned in Section 3, the problem in Figure 10 can be reduced into merging two rigid bodies $p_1p_2p_3p_4$ and $p_5p_6p_7p_8$. We take $P_5P_6P_7P_8$ as the the base and $p_1p_2p_3p_4$ the platform. The constraints are $|l_1p_4| = 0$, $|l_2p_3| = 0$ and $|p_5l_3| = 0$, which is an* **LPP-PLL** *case. Let $p_7 = (0, 0)$. The parametric equations for lines l_1, l_2 are $p = (0, 0) + u_1(0, 1)$ and $p = (0, 0) + u_2(1, 0)$. Let point p_3 be the origin of the moving coordinate system. Then $p_3 = (x_3, x_4)$. Let $|p_6p_7| = b_2$, $|p_5p_6| = b_3$ and $|p_3p_4| = h_3$. Thus the coordinates for points p_4 and p_5 are $p_4 = (-x_2h_3 + x_3, x_1h_3 + x_4)$ and $p_5 = (b_2, b_3)$. The parametric equation of line l_3 is $p = (x_3, x_4) + u_3(x_1, x_2)$. The equation system is*

$$\begin{cases} x_1^2 + x_2^2 - 1 = 0 \\ |x_2(b_2 - x_3) - x_1(b_3 - x_4)| = 0 \\ | - h_3x_2 + x_3| = 0 \\ |x_4| = 0 \end{cases} \tag{6}$$

The triangular form is

$$\begin{cases} b_2x_3 - h_3x_1b_3 + h_3^2x_1^2 - h_3^2 = 0 \\ x_2b_2 - x_1b_3 + h_3x_1^2 - h_3 = 0 \\ h_3^2x_1^4 - 2x_1^3b_3h_3 + (b_3^2 + b_2^2 - 2h_3^2)x_1^2 \\ \qquad +2h_3x_1b_3 - b_2^2 + h_3^2 = 0 \\ x_4 = 0 \end{cases} \tag{7}$$

It has 4 solutions at most and is not ruler and compass constructible.

4.2.3 Analytic Solutions to GSP in 2D

A problem is called RC-constructible if it can be constructed with ruler and compass. The definition for RC-constructible can be found in [8]. When $\mathrm{CN}(\mathcal{B},\mathcal{U}) = 3$, the problem can be classified into eleven different cases:

1. The case **dda** is RC-constructible.
2. PPP-LLL and LLL-PPP are reduced to solving of one quadratic and three linear equations, and are RC-constructible.
3. PPP-LLP, LLP-PPP, LPP-PLL and LLP-PPL are reduced to solving of one quantic and three linear equations, and are not be RC-constructible. We use the method in [8] to decide that the roots of these equations can not be constructed with ruler and compass.
4. PPP-LPP, LPP-PPP, LPP-PLP and PPP-PPP are reduced to solving of one equation of degree six and three linear equations, which are not RC-constructible. The polynomials of degree six in these cases are irreducible. Then it is obvious that the roots of these equations can not be constructed with ruler and compass.

5 Solving of General Construction Sequences in 3D

5.1 Classification of GC in 3D

Use the notations introduced at the beginning of Section 4.

Theorem 5.1 *Let V_3 be the set of points and planes on the dependent object $\mathcal{U}$, V_4 the set of lines on $\mathcal{U}$, and $V = V_3 \cup V_4$. We have $3 \le CN(\mathcal{B},\mathcal{U}) \le DOF(V) - |V| = 2|V_3| + 3|V_4|$.*

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 $\mathrm{CN}(\mathcal{B},\mathcal{U}) \ge 3$. From [27], $\mathcal{U}_i$ can be changed to a strongly connected directed graph. From the definition of GC, $\mathcal{U}_i$ satisfies the following conditions [27].

1. If $|V| > 1$, for every vertex v in V there exists at least one constraint between v and a primitive in V, i.e. $\mathrm{DEG}(v) > 1$. Otherwise, v can be determined by $\mathcal{B}$ alone, which contradicts to the minimal property of V (the third condition in the definition of GC).
2. $\mathcal{U}_i$ 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 $|V|$ constraints, i.e. $\mathrm{DOC}(V) \ge |V|$. Since both $\mathcal{B}$ and $\mathcal{B} \cup \mathcal{U}_i$ are rigid bodies, we need exactly $\mathrm{DOF}(V) = 3|V_3| + 4|V_4|$ constraints to determine $\mathcal{U}_i$. In other words, we have

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

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

The computation of $\mathcal{U}$ with respect to $\mathcal{B}$ can be divided into the following cases.

1. If CN($\mathcal{B}$,$\mathcal{U}$) $= 3$, $\mathcal{U}$ consists of a point or a plane, which can be constructed explicitly.

2. If CN($\mathcal{B}$,$\mathcal{U}$) $= 4$, $\mathcal{U}$ consists of a line, which can be constructed explicitly.

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

4. If CN($\mathcal{B}$,$\mathcal{U}$) $= 6$, there exist DOF$(\mathcal{U}) - 6$ constraints between primitives in $\mathcal{U}$. Hence $\mathcal{U}$ is a rigid body. It may be considered as to assembly two rigid bodies 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 $\mathcal{U}$.

For the first three cases of merge patterns, the solution is relatively easy. The fourth and fifth cases could be very difficult. Usually, analytical solutions are too large to be computed. We may use numerical computation to find some of the solutions. Techniques from AI could be used to simplify them further [4].

In the above, we only concern the structure of the merge patterns. It could happen that a structurally well-constrained problem could 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 the basic merge pattern $(\mathcal{B},\mathcal{U})$, we say that it is an *angular conflict* pattern.

1. A plane or a line in $\mathcal{U}$ has more than two angular constraints with elements in $\mathcal{B}$.

2. If CN($\mathcal{B}$,$\mathcal{U}$) $= 6$ and there are more than three angular constraints between $\mathcal{B}$ and $\mathcal{U}$.

3. 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 5.2 *In general, an angular conflict pattern cannot be realized in the Euclidean space.*

Proof. Since a line or a plane has two angular (directional) degrees of freedom and a rigid body has three angular degrees of freedom, 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. If the third case occurs, let H be the set of lines and planes in $\mathcal{U}$. All the more than $2(l + h)$ angular constraints are for primitives in H. But, H has at most $2(l + h)$ angular degrees of freedom, which will lead to angular conflicts in the general case.

5.2 Generalized Stewart Platform

Let ($\mathcal{B}$,$\mathcal{U}$) be a basic merge 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.

4D2A: There are four distance and two angular constraints.

5D1A: There are five distance and one angular constraints.

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* [2], which is a **6D** problem where all distance constraints are between points.

This platform is extensively studied because it has many important applications. For a survey, please consult [2]. 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 geometric constraint solving.

The system $\mathcal{B},\mathcal{U}$ satisfying CN($\mathcal{B}$,$\mathcal{U}$) $= 6$ will be called a *generalized Stewart platform* (GSP). Figure 11 is the illustration of GSP in 3D.

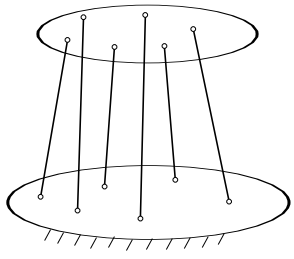

Figure 11: A Stewart Platform in 3D

We could simplify a GSP ($\mathcal{B}$,$\mathcal{U}$) as follows. We may solve the rigid body $\mathcal{U}$ separately with Algorithm 3.2. Let

$\mathcal{U}'$ be the set of vertices in $\mathcal{U}$, such that each vertex in $\mathcal{U}'$ has a constraint with a vertex in $\mathcal{B}$. Since $\mathcal{U}$ is a rigid body, we may add a reasonable number of constraints to $\mathcal{U}'$ so that $\mathcal{U}'$ becomes a rigid body. Then the basic merge pattern $(\mathcal{B},\mathcal{U})$ could be simplified to $(\mathcal{B},\mathcal{U}')$. As a consequence, we may assume that $|\mathcal{U}| \leq 6$. For instance, the basic pattern in Figure 12(a) could be reduced to the one in Figure 12(b).

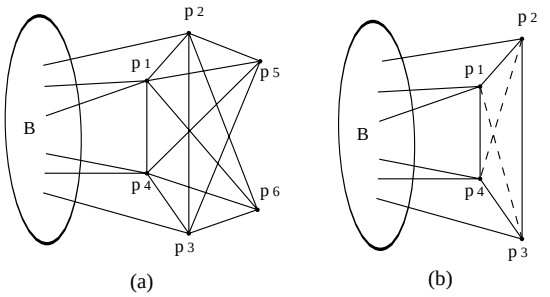

Figure 12: A GSP and its simplified form

To find the analytic solutions to the generalized Stewart platforms is a difficult problem and has been studied extensively [2]. Most of the work is focused on the original Stewart platform, where all the primitives are points. It is proved that this platform generally has forty solutions [2]. But, to our knowledge the analytic solutions to the Stewart platform are not found.

5.3 Analytic Solutions to 3D3A GSP

In this section, we try to give the analytical solution to the **3D3A** Stewart platform. This platform is easier because we may impose the angular constraints first and then the distance constraints.

It is clear that rotational constraints can violate previous imposed distance constraints while a translation does not violate previous imposed angular constraints. Therefor, we impose the three angular constraints first to remove three rotational degrees of freedom, and then determine all the degrees of freedom by imposing the three distance constraints[24].

Let us state the problem precisely as follows: *$\mathcal{B}$ is a rigid body whose position is fixed. $\mathcal{U}$ is a rigid body with three distance and three angular constraints with $\mathcal{B}$. We need to position the rigid body $\mathcal{U}$ relative to the rigid body $\mathcal{B}$.*

In what below, assume that ($\mathcal{B}$,$\mathcal{U}$) is a GSP with base $\mathcal{B}$ and platform $\mathcal{U}$.

Imposing Angular Constraints. The angular constraints and their valences in 3D are listed in Table 1 [24].

Constraint type	Geometric entities	Valence
angle	line, line	1
	line, plane	1
	plane, plane	1
parallelism	line, line	2
	line, plane	2
	plane, plane	2

Table 1. Valencies of the angular constraints

There are in fact only two types of angular constraints. One is the angular constraint of valency 1 and the other is the parallelism constraint of valency 2. There are only two cases to remove the three directional degrees of freedom: imposing a parallelism constraint of valency 2 and an angular constraint of valency 1; imposing three angular constraints of valency 1.

It is obvious that the first case can be reduced to solving two linear and one quadratic equation. Therefore, the problem could have two solutions.

Now we consider the case of imposing three angular constraints of valency 1. Since a vector can be used to represent both the orientation of a line and the normal of a plane, we only consider angular constraints between two vectors[24]. Let $\mathbf{l_{11}}$, $\mathbf{l_{12}}$ and $\mathbf{l_{13}}$ be three lines in $\mathcal{B}$, the base set, and $\mathbf{l_{21}}$, $\mathbf{l_{22}}$ and $\mathbf{l_{23}}$ three lines in $\mathcal{U}$, the platform set. Let the parametric equations of line $\mathbf{l_{1i}}$ be $\mathbf{p} = \mathbf{p_{1i}} + u_{1i}\mathbf{s_{1i}}$, where $\mathbf{s_{1i}} = (l_i, m_i, n_i)$, $|\mathbf{s_{1i}}| = 1$, $i = 1, 2, 3$. Assume that after imposing three angular constraints the parametric equations of the three lines in the platform are $\mathbf{l_{2i}}$ $\mathbf{p} = \mathbf{p_{2i}} + u_{2i}\mathbf{s_{2i}}$, where $\mathbf{s_{2i}} = (x_i, y_i, z_i)$, $|\mathbf{s_{2i}}| = 1$, $i = 1, 2, 3$.

Let the three angular constraints be ANG($\mathbf{l_{11}}, \mathbf{l_{21}}$) $= \alpha_1$, ANG($\mathbf{l_{12}}, \mathbf{l_{22}}$) $= \alpha_2$, ANG($\mathbf{l_{13}}, \mathbf{l_{23}}$) $= \alpha_3$ and $d_i = \cos\alpha_i (i = 1, 2, 3)$. Since $\mathcal{U}$ is a rigid body, the angles between three lines in $\mathcal{U}$ are also known: ANG($\mathbf{l_{21}}, \mathbf{l_{22}}$) $= \beta_1$, ANG($\mathbf{l_{21}}, \mathbf{l_{23}}$) $= \beta_2$, ANG($\mathbf{l_{22}}, \mathbf{l_{23}}$) $= \beta_3$. Let $\cos\beta_j = d_{j+3} (j = 1, 2, 3)$. We need to determine three unit vectors $\mathbf{s_{21}}$, $\mathbf{s_{22}}$, $\mathbf{s_{23}}$ by solving the following nine equations

$$\begin{cases} l_1x_1 + m_1y_1 + n_1z_1 - d_1 = 0 \\ l_2x_2 + m_2y_2 + n_2z_2 - d_2 = 0 \\ l_3x_3 + m_3y_3 + n_3z_3 - d_3 = 0 \\ x_1x_2 + y_1y_2 + z_1z_2 - d_4 = 0 \\ x_1x_3 + y_1y_3 + z_1z_3 - d_5 = 0 \\ x_2x_3 + y_2y_3 + z_2z_3 - d_6 = 0 \\ x_1^2 + y_1^2 + z_1^2 - 1 = 0 \\ x_2^2 + y_2^2 + z_2^2 - 1 = 0 \\ x_3^2 + y_3^2 + z_3^2 - 1 = 0 \end{cases} \tag{8}$$

Because a line has two rotational degrees of freedom, the problem can be classified into three cases shown in Figure 13.

For the case in Figure 13-(a), we could set $(l_1, m_1, n_1) = (0, 0, 1)$ and $m_2 = 0$. For the case in Figure 13-(b), we could set $(l_1, m_1, n_1) = (0, 0, 1)$ and $m_3 = 0$. With Wu-Ritt's

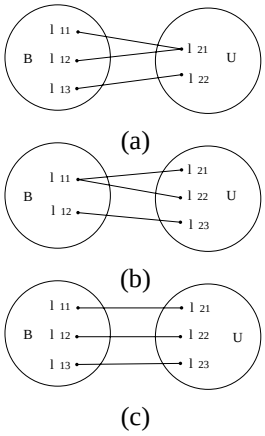

Figure 13: Three cases of three angle constraints

method [36], we could reduce the equation system (8) into the following triangular form.

$$\begin{cases} l_2x_1 + z_{111} = 0 \\ z_{121}x_2 + (z_{122}y_1 + z123) * y2 + z_{124} = 0 \\ z_{131}z_2 + (z_{132}y_1 + z_{133})y_2 + z_{134} = 0 \\ (z_{141}y_1 + z_{142})y_2^2 + (z_{143}y_1 + z_{144})y_2 + z_{145} = 0 \\ l_2^2y_1^2 + z_{151} = 0 \end{cases} \tag{9}$$

The above equation system has three linear and two quadratic equations and has at most four solutions. Other cases can be treated similarly.

For the case shown in Figure 13-(c), it is difficult to transform the corresponding equation system into triangular form. But, we may compute the m-Bezout number for the system as follows [32]. Let $T_1 = \{x_1, y_1, z_1\}, T_2 = \{x_2, y_2, z_2\}, T_3 = \{x_3, y_3, z_3\}$. The degree vectors of the nine equations in (8) for T_1, T_2, T_3 are $(1,0,0)$, $(0,1,0)$, $(0,0,1)$, $(1,1,0)$, $(1,0,1)$, $(0,1,1)$, $(2,0,0)$, $(0,2,0)$, $(0,0,2)$. Then the m–Bezout number is the coefficient of $T_1^3T_2^3T_3^3$ in the polynomial $T_1T_2T_3(T_1+T_2)(T_1+T_3)(T_2+T_3)(2T_1)(2T_2)(2T_3)$, which is 16. As a consequence, we know that the above equation system has at most 16 solutions.

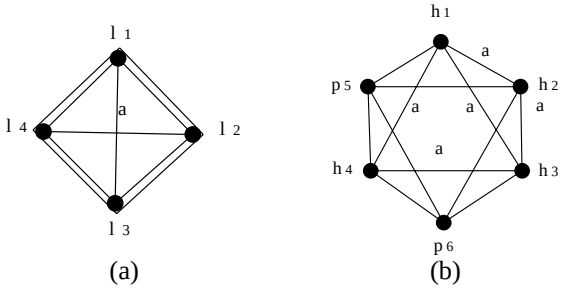

Figure 14: Examples for case 3D3A

Example 5.3 *The constraint graph shown in Figure 14-(a) is a basic configuration with four lines from [9]. Lines labelled by a represent angular constraints. If there are two lines between two vertices, one line represents an angular and the other represents a distance constraint. Let l_3l_4 be the base and l_1l_2 the platform. Imposing three angular constraints to l_1l_2 is just corresponding to the case shown in Figure 13-(a). So there are four solutions at most.*

The constraint graph shown in Figure 14-(b) is a basic configuration from [5]. $\mathbf{h_i}(i = 1, \cdots, 4)$ are planes and $\mathbf{p_5}$ and $\mathbf{p_6}$ are points. Let triangle $h_3h_4p_6$ be the base and $h_1h_2p_5$ the dependent . The three angular constraints are imposed on h_1, h_1, h_2. This problem corresponds to the case shown in Figure 13-(a). So there are four solutions at most.

Imposing Distance Constraints. The distance constraints and their valences in 3D are listed in Table 2 [24].

Constraint type	Geometric entities	Valence
distance	point, point	1
	point, line	1
	point, plane	1
	line, line	1
	line, plane	1
	plane, plane	1
coincidence (distance is zero)	point, point	3
	point, line	2
	line, line	1
	point, plane	1

Table 2. Valencies of the distance constraints

There are two ways to remove the three translational degrees of freedom: (1) imposing a distance constraint of valency two, which is a point-line coincident and a distance constraint of valency one; (2) imposing three distance constraints of valency one. We discuss the second case first. The problem of imposing three distance constraints of valency one can be classified into five cases shown in Figure 15.

Type	Geo. Prim	Parametric equation for translation space
LL	plane	$\mathbf{R}^p(u,v) = \mathbf{p_l} + r_l\mathbf{m} + u\mathbf{I_1} + v\mathbf{I_2}$
PH	plane	$\mathbf{R}^p(u,v) = \mathbf{p_0} + r_p\mathbf{m} + u\mathbf{a} + v\mathbf{b}$
PP	sphere	$\mathbf{R}^s(\phi,\theta) = \mathbf{C_0} + r_s(\sin\phi\cos\theta\mathbf{i} + \sin\phi\sin\theta\mathbf{j})$
PL	cylinder	$\mathbf{R}^c(\rho,\phi) = \mathbf{p_a} + \rho\mathbf{I} + r_\rho(\cos\phi\mathbf{m} + \sin\phi\mathbf{n})$

Table 3 Parametric equations for translations space

After removing three directional degrees of freedom for U, there are four basic types of translation spaces, shown in Table 3, where **LL** means that the distance constraint is between two lines; **PH** means that the distance constraint is between a point and a plane, etc. In case **LL**, assuming that line $l_1 \in \mathcal{B}$ and line $\mathbf{l_2} \in \mathcal{U}$, $\mathbf{p_l}$ is a point on l_1. Then $\mathbf{I_1}$ and $\mathbf{I_2}$ are unit vectors parallel to l_1 and l_2 respectively; $\mathbf{m}$ is a unit vector perpendicular to $\mathbf{l_1}$ and l_2; and r_l is the distance. In case **PH**, if the point is in $\mathcal{B}$, then $\mathbf{p_0}$ is the corresponding

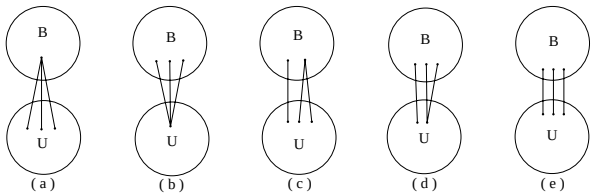

Figure 15: Five Cases of Three distance Constraints

point. Otherwise it is a point on the plane. **a** and **b** are mutually perpendicular unit vectors parallel to this plane; **m** is a unit vector perpendicular to **a** and **b**; and r_p is the distance. In case **PP**, $\mathbf{C_0}$is a point in $\mathcal{B}$ and r_s is the distance. In case **PL**[24], **I** is a unit vector parallel to the line and **m** and **n** are mutually perpendicular unit vectors perpendicular to **I**. If the point is in the base , $\mathbf{p_a}$ is the corresponding point. Otherwise it is a point on the line.

We can always assume that the primitives in $\mathcal{U}$ are three points. If the given vertex in the platform is a plane or a line, we can take a point on the plane or the line. Proofs for the correctness of this are omitted. Let $\mathbf{p_1}$, $\mathbf{p_2}$ and $\mathbf{p_3}$ be the points in $\mathcal{U}$, d_1, d_2 and d_3 the corresponding distances. The translation spaces of constraints d_1, d_2 and d_3 are $\mathbf{R}^{d_1}(u_1)$, $\mathbf{R}^{d_2}(u_2)$ and $\mathbf{R}^{d_3}(u_3)$ respectively. Imposing constraints d_1, d_2 and d_3 means that we must find points $\mathbf{p_1}^*$ in $\mathbf{R}^{d_1}(u_1)$, $\mathbf{p_2}^*$ in $\mathbf{R}^{d_2}(u_2)$ and $\mathbf{p_3}^*$ in $\mathbf{R}^{d_3}(u_3)$, such that segments $\mathbf{p_1}^*\mathbf{p_3}^*$ and $\mathbf{p_2}^*\mathbf{p_3}^*$ are parallel and equal to segments $\mathbf{p_1p_3}$ and $\mathbf{p_2p_3}$, respectively. The parametric equation of line $\mathbf{p_1^*p_3^*}$ is $\mathbf{p} = \mathbf{p_1}^* + v_1\mathbf{s_1}$, where v_1 is a free parameter and $\mathbf{s_1}$ is a unit vector. The parametric equation of line $\mathbf{p_2^*p_3^*}$ is $\mathbf{p} = \mathbf{p_2}^* + v_2\mathbf{s_2}$, where v_2 is a free parameter and $\mathbf{s_2}$ is a unit vector. Let $t_1 = |\mathbf{p_1p_3}|$ and $t_2 = |\mathbf{p_2p_3}|$. Then we have $\mathbf{p_3}^* = \mathbf{p_1}^* + t_1\mathbf{s_1}$ and $\mathbf{p_3}^* = \mathbf{p_2}^* + t_2\mathbf{s_2}$.

After obtaining the analytical solutions to point $\mathbf{p_3^*}$, we can translate $\mathcal{U}$ along the vector $\mathbf{t} = \mathbf{p_3^*} - \mathbf{p_3}$ and get the analytic solutions to point $\mathbf{p_1^*}$ according to $\mathbf{p_3^*} = \mathbf{p_1^*} + t_1\mathbf{s_1}$, point $\mathbf{p_2^*}$ according to $\mathbf{p_3^*} = \mathbf{p_2^*} + t_2\mathbf{s_2}$, and hence the position of $\mathcal{U}$.

According to theorem 4.3, it's obvious that point $\mathbf{p_3}^*$ satisfies the following equations

$$\begin{cases} \mathbf{p} = \mathbf{R}^{d_1}(u_1) + t_1\mathbf{s_1} \\ \mathbf{p} = \mathbf{R}^{d_2}(u_2) + t_2\mathbf{s_2} \\ \mathbf{p} = \mathbf{R}^{d_3}(u_3) \end{cases} \tag{10}$$

1. If $t_1 \neq 0$ and $t_2 \neq 0$ in (10), that is $\mathbf{p_1}$, $\mathbf{p_2}$ and $\mathbf{p_3}$ are three different points, equation system (10) corresponds to cases (a), (c) and (e) in Figure 15.
2. If $t_1 = 0$ and $t_2 = 0$, that is $\mathbf{p_1} = \mathbf{p_2} = \mathbf{p_3}$, equation system (10) corresponds to cases (b) in Figure 15.
3. If $t_1 = 0$ and $t_2 \neq 0$, that is point $\mathbf{p_1} = \mathbf{p_2}$ and $\mathbf{p_3} \neq \mathbf{p_1}$, equation system (10) corresponds to case (d) shown in Figure 15.

Since we'll use implicit equations during computation, in what below, we will list the implicit equations for $\mathbf{R}^d(u)+t\mathbf{s}$, where $\mathbf{R}^d(u)$ is the translation space in Table 1 and $t\mathbf{s}$ is a constant vector.

Case **LL:** The implicit equation is $(m_2n_1-n_2m_1)(x-x_1-r_ll_3)+(l_1n_2-n_1l_2)(y-y_1-r_lm_3)+(m_1l_2-l_1m_2)(z-z_1-r_ln_3) = 0$, where $\mathbf{p} = (x, y, z)$, $\mathbf{p_l} = (x_{p_l}, y_{p_l}, z_{p_l})$, $\mathbf{s} = (l_s, m_s, n_s)$, $\mathbf{I_1} = (l_1, m_1, n_1)$, $\mathbf{I_2} = (l_2, m_2, n_2)$, $\mathbf{m} = (l_3, m_3, n_3)$, $\mathbf{m} = \pm\mathbf{I_1}\times\mathbf{I_2}$, $x_1 = x_{p_l}+tl_s$, $y_1 = y_{p_l}+tm_s$, $z_1 = z_{p_l} + tn_s$.

The implicit equation can be simplified as $d_1x + d_2y + d_3z + d4 = 0$ where $d_1 = (m_2n_1 - n_2m_1)$, $d_2 = (l_1n_2 - n_1l_2)$, $d_3 = (m_1l_2 - l_1m_2)$, $d_4 = -(m_2n_1 - n_2m_1)(x_1 + r_ll_3)-(l_1n_2-n_1l_2)(y_1+r_lm_3)-(m_1l_2-l_1m_2)(z_1+r_ln_3)$. This is a plane.

Case **PH:** The implicit equation is $(m_2n_1-n_2m_1)(x-x_1-r_pm_3)+(l_1n_2-n_1l_2)(y-y_1-r_pm_3)+(m_1l_2-l_1m_2)(z-z_1-r_pn_3) = 0$, where $\mathbf{p} = (x, y, z)$, $\mathbf{p_0} = (x_{p_0}, y_{p_0}, z_{p_0})$, $\mathbf{s} = (l_s, m_s, n_s)$, $\mathbf{a} = (l_1, m_1, n_1)$, $\mathbf{b} = (l_2, m_2, n_2)$, $\mathbf{m} = (l_3, m_3, n_3)$, $\mathbf{m} = \pm\mathbf{a}\times\mathbf{b}$, $x_1 = x_{p_0}+tl_s$, $y_1 = y_{p_0}+tm_s$, $z_1 = z_{p_0} + tn_s$.

The implicit equation can be simplified as $d_1x + d_2y + d_3z + d4 = 0$ where $d_1 = (m_2n_1 - n_2m_1)$, $d_2 = (l_1n_2 - n_1l_2)$, $d_3 = (m_1l_2 - l_1m_2)$, $d_4 = -(m_2n_1 - n_2m_1)(x_1 + r_pm_3) - (l_1n_2 - n_1l_2)(y_1 + r_pm_3) - (m_1l_2 - l_1m_2)(z_1 + r_pn_3)$. This is a plane.

Case **PP:** The implicit equation is $(x-x_1)^2+(y-y_1)^2+(z-z_1)^2 - r_s^2 = 0$, where $\mathbf{p} = (x, y, z)$, $\mathbf{C_0} = (x_{\mathbf{C_0}}, y_{\mathbf{C_0}}, z_{\mathbf{C_0}})$, $\mathbf{s} = (l_s, m_s, n_s)$, $x_1 = x_{C_0} + tl_s$, $y_1 = y_{C_0} + tm_s$, $z_1 = z_{C_0} + tn_s$.

The implicit equation can be simplified as $x^2+y^2+z^2+d_1x + d_2y + d_3z + d_4 = 0$ where $d_1 = -2x_1$, $d_2 = -2y_1$, $d_3 = -2z_1$, $d_4 = x_1^2 + y_1^2 + z_1^2 - r_s^2$. This is a sphere.

Case **PL:** The implicit equation is $(x-x_1-\rho l)^2+(y-y_1-\rho m)^2 + (z - z_1 - \rho n)^2 - r_\rho^2 = 0$, where
$\rho = \frac{(m_1n_2-n_1m_2)x+(n_1l_2-l_1n_2)y+(l_1m_2-m_1l_2)z}{(l_1m_2-n_1m_2-m_1l_2+m_1n_2)l+(n_1l_2-l_1n_2)m} + \frac{(n_1m_2-n_2m_1)x_1+(n_2l_1-l_2n_1)y_1+(l_2m_1-l_1m_2)z_1}{(l_1m_2-n_1m_2-m_1l_2+m_1n_2)l+(n_1l_2-l_1n_2)m}$,
$\mathbf{p} = (x, y, z)$, $\mathbf{p_a} = (x_{\mathbf{p_a}}, y_{\mathbf{p_a}}, z_{\mathbf{p_a}})$, $\mathbf{s} = (l_s, m_s, n_s)$, $\mathbf{I} = (l, m, n)$, $\mathbf{m} = (l_1, m_1, n_1)$, $\mathbf{n} = (l_2, m_2, n_2)$, $x_1 = x_{\mathbf{p_a}} + tl_s$, $y_1 = y_{\mathbf{p_a}} + tm_s$, $z_1 = z_{\mathbf{p_a}} + tn_s$.

The implicit equation can be simplified as

$$d_1x^2+d_1y^2+d_1z^2+(d_2y+d_3z+d_4)x+(d_5z+d_6)y+d_7z+d_8 = 0$$

where the d_i are some constants. The implicit equation represents a cylinder.

From these equations, we see that the translation spaces are planes, spheres or cylinders which are denoted by **P**, **S** and **C**. For instance, **PPS** means that we need to find the intersection points of two planes and one sphere. To satisfy three distance constraints can be classified into the following cases, which are all solved with Wu-Ritt's method [36]. For

the explicit formulas, please refer to

http://www.mmrc.iss.ac.cn/~xgao/publ.html.

1. The equation system for **PPP** consists of three linear equations. Then it has one solution.
2. The equation system for **PPC** or **PPS** consists of two linear and one quadratic equation. Then they have two solutions at most.
3. The equation system for **SSS** or **PSS** can be reduced into a triangular set consisting of one quadratic and two linear equations. Then they have two solutions at most.
4. The equation system for **PCC**, **SSC** or **PSC** can be reduced into a triangular set consisting of one quartic and two linear equations. Then they have four solutions at most.
5. The equation system for **CCC** or **SCC** can be reduced into a triangular set consisting of a degree eight equation and two linear equations. They have eight solutions at most.

In what below, we illustrate the solving process by giving the computation procedure for case **CCC**, i.e. to find the intersection points of three cylinders. By making a proper coordinate system, the equation system can be represented as

$$\begin{cases} x^2+y^2-d_1=0 \\ f_1x^2+f_1y^2+f_1z^2+(f_2y+f_3z+f_4)x \\ \quad +(f_5z+f_6)y+f_7z+f_8=0 \\ g_1x^2+g_1y^2+g_1z^2+(g_2y+g_3z+g_4)x \\ \quad +(g_5z+g_6)y+g_7z+g_8=0 \end{cases} \tag{11}$$

where the f_i and g_i are constants. The above equation system can be reduced into the following triangular form which consists of a degree eight equation and two linear equations.

$$\begin{cases} (z_{611}x^4+z_{612}x^3+z_{613}x^2+z_{614}x+z_{615})z+z_{616}x^5 \\ \quad +z_{617}z^4+z_{618}z^3+z_{619}z^2+z_{6110}z+z_{6111}=0 \\ (z_{621}x^3+z_{622}x^2+z_{623}x+z_{624})y+z_{625}x^4+z_{626}x^3 \\ \quad +z_{627}x^2+z_{628}x+z_{629}=0 \\ z_{631}x^8+z_{632}x^7+z_{633}x^6+z_{634}x^5+z_{635}x^4+z_{636}x^3 \\ \quad +z_{637}x^2+z_{638}x+z_{639}=0. \end{cases} \tag{12}$$

Example 5.4 *Continue from Example 5.3. Now impose three distance constraints to the problems in Figure 14, respectively. It is obviously that each equation system consists of three linear equations, and only has one solution. Thus each problem shown in Figure 14 has four solutions at most. For the problem in Figure 14(a), this result is better than that in [9]. For the problem in Figure 14(b), this is the same as that in [5].*

For the case of imposing a distance constraint of valency 2 and a distance constraint of valency 1, it is obvious that the corresponding equations for both line-line coincident and point-line coincident are two linear equations. From the above discussion, we know the equation for the distance constraint of valency one is linear or quadratic after removing three rotational degrees. So this case has one or two solutions.

Theorem 5.5 *We generally could have 2, 4, 16 solutions when imposing the angular constraints. We generally could have 1,2,4, 8 solutions when imposing the distance constraints. Therefore, the 3D3A GSP generally could have* $2^k, k=1,\cdots,7$ *solutions depending on the types of the constraints imposed on it.*

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.

Our decomposition procedure for a constraint problem is shown in Figure 16. First, a constraint problem is reduced to several GCs with a c-tree decomposition; a GC is reduced to basic merging patterns; finally, a basic merging patten may have three possible forms: explicit construction, GSP, or general form. Solution to explicit construction is generally easy. Solution of GSP is discussed in section 4.2 and 5.3.

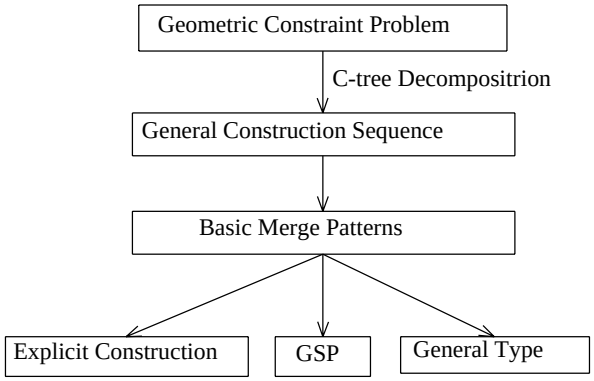

Figure 16: Solving a Constraint Problem

Solution of general basic merging pattern could be 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 such as the one in [12].

We have implemented the c-tree decomposition method in our software package MMP/Geometer (http://www.mmrc.iss.ac.cn/ mmsoft). 5solved with the software.

References

[1] B. Brüderlin, Using Geometric Rewriting Rules for Solving Geometric Problems Symbolically, *Theoretical Computer Science*, **116**, 291-303, 1993.

[2] B. Dasgupta and T. S. Mruthyunjaya, The Stewart Platform Manipulator: A Review, *Mechanism and Machine Theorey*. **35**, 15-40, 2000.

[3] J. X. Dong, J. X. Ge, Y. Gao and H. L. Li, A New Method for Constraint Satisfaction in Parametric Drafting Systems, *Journal of CAD & CG*, **9**(6), 513-519, 1997

[4] J. F. Dufourd, P. Mathis and P. Schreck, Geometric Construction by Assembling Solved Subfigures, *Artificial Intelligence*, **99**, 73-119, 1998.

[5] C. Durand and C. M. Hoffmann, A Systematic Framework for Solving Geometric Constraints Analytically, *J. of Symbolic Computation*, **30**(5), 493-529, 2000.

[6] I. Fudos and C.M. Hoffmann, A Graph-Constructive Approach to Solving Systems of Geometric Constraints, *ACM Trac. on Graphics*, **16**(2), 179-216, 1997.

[7] X. S. Gao and S. C. Chou, Solving Geometric Constraint Systems I. A Global Propagation Approach, *Computer Aided Design*, **30**(1), 47-54, 1998.

[8] X. S. Gao and S. C. Chou, Solving Geometric Constraint Systems II. A Symbolic Approach and Decision of RC-constructibility, *Computer Aided Design*, **30**(2), 115-122, 1998.

[9] X. S. Gao, C. M. Hoffmann and W. Q. Yang, Solving Basic Geometric Constraint Configurations with Locus Intersection, *Proc. ACM SM02*, 95-104, ACM Press, New York, 2002.

[10] X.-S. Gao and G. Zhang, Geometric Constraint Solving via C-tree Decomposition, *ACM SM03*, 45-55, Seattle, USA, ACM Press, New York, 2003.

[11] X.S. Gao and G. Zhang, Classification and Solving of Merge Patterns in Geometric Constraint Solving, *Proc. Shape Modeling and Applications*, 89-90, IEEE press, 2003.

[12] J. Ge, S.C. Chou and X. Gao, Geometric Constraint Satisfaction Using Optimization Methods, *Computer Aided Design*, **31**(14), 867-879, 2000.

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

[14] C. Jermann, B. Neveu and G. Trombettoni, On the Structural Rigidity for GCSPs, talk at *ADG2002*, RISC, Linz, Austria, 2003.

[15] C. M. Hoffmann and P. J. Vermeer, Geometric Constraint Solving in R^2 and R^3, in *Computing in Euclidean Geometry*, D. Z. Du and F. Huang (eds), World Scientific, Singapore, 266-298, 1995

[16] C. M. Hoffmann, A. Lomonosov and M. Sitharam, Finding Solvable Subsets of Constraint Graphs, in *LNCS*, NO. 1330, Springer, Berlin Heidelberg, 163-197, 1997.

[17] C. M. Hoffmann, A. Lomonosov and M. Sitharam, Decomposition Plans for Geometric Constraint Systems, I: Performance Measures for CAD, **31**, 367-408; II: New Algorithms, **31**, 409-427, *J. of Symbolic Computation*, 2001.

[18] R. Joan-Arinyo and A. Soto, A Correct Rule-Based Geometric Constraint Solver, *Computers and Graphics*, **21**(5), 599-609, 1997.

[19] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta and J. Vilaplana-Pastó, Revisiting Decomposition Analysis of Geometric Constraint Graphs, *Proc. ACM SM02*, 105-115, ACM Press, New York, 2002.

[20] 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*, 33-44, ACM Press, New York, 2003.

[21] A. Kanevsky and V. Ramachandran, Improved Algorithms for Graph Four-connectivity, *Proc. 28th IEEE Symp. Foundations of Comp. Sci.*, Los Angeles, 252-259, 1987.

[22] K. Kondo, Algebraic Method for Manipulation of Dimensional Relationships in Geometric Models, *Computer Aided Design*, **24**(3), 141-147, 1992.

[23] G. A. Kramer, Solving Geometric Constraints Systems: A Case Study in Kinematics, MIT Press, Cambridge Massachusetts, 1992.

[24] A. V. Kumar and L. Yu, Sequential Constraint Imposition for Dimension-driven Solid Models, *Computer Aided Design*, **33**, 475-486, 2001

[25] H. Lamure and D. Michelucci, Solving Geometric Constraints by Homotopy, *IEEE Trans on Visualization and Computer Graphics*, **2**(1):28-34, 1996.

[26] H. Lamure and D. Michelucci, Qualitative Study of Geometric Constraints, in *Geometric Constraint Solving and Applications*, 234-258, Springer, Berlin, 1998.

[27] R. S. Latham and A. E. Middleditch, Connectivity Analysis: A Tool for Processing Geometric Constraints, *Computer Aided Design*, **28**, 917-928, 1994.

[28] J. Y. Lee and K. Kim, Geometric Reasoning for Knowledge-Based Design Using Graph Representation, *Computer Aided Design*, **28**(10), 831-841, 1996.

[29] Y. T. Li, S. M. Hu and J. G Hu, Hybrid Model of Geometric Constraint Satisfaction, *Journal of Computer Research and Development*, **37**(10), 1233-1239, 2000

[30] V. C. Lin, D. C. Gossard and R. A. Light, Variational Geometry in Computer-Aided Design, *Computer Graphics*, **15**(3), 171-177, 1981.

[31] H. L. Liu, T. Z. Zhang and H. S. Ding, Forward Solution of the 3-RPR Planar parallel Mechanism with Wu's Method, *Journal of Beijing Institute of Technology*, **20**(5), 565-569, 2000

[32] A. Morgan and A. Sommese, A Homotopy for Solving General Polynomial Systems that Respect m-homogeneous Structures, *Appl. Math. Comput.*, **24**, 95-114, 1987.

[33] J.C. Owen, Algebraic Solution for Geometry from Dimensional Constraints, in *ACM Symp., Found of Solid Modeling*, 397-407, ACM Press, New York, 1991.

[34] J.C. Owen, Constraints on Simple Geometry in Two and Three Dimensions, *Int. J. of Computational Geometry and Applications*, 6(4), 421–434, 1996.

[35] A. Verroust, F. Schonek and D. Roller, Rule-Oriented Method for Parameterized Computer-Aided Design, *Computer Aided Design*, **24**(10), 531-540, 1992.

[36] W.T. Wu, Basic Principles of Mechanical Theorem Proving in Geometries, Science Press, Beijing, 1984; English Translation, Springer, Wien, 1994.