Computer-Aided Design 36 (2004) 111–122

COMPUTER-AIDED DESIGN

www.elsevier.com/locate/cad

Solving spatial basic geometric constraint configurations with locus intersection

Xiao-Shan Gao[a,1], Christoph M. Hoffmann[b,*,2], Wei-Qiang Yang[b]

[a]Institute of Systems Science, AMSS, Academia Sinica, Beijing 100080, People's Republic of China
[b]Department of Computer Science, Purdue University, West Lafayette, IN 47907-1398, USA

Abstract

A basic idea of geometric constraint solving (GCS) is to decompose the constraint problem into smaller ones according to some basic configurations. In this paper, we find all spatial basic configurations involving points, lines, and planes containing up to six geometric primitives in an automated way. Many of these basic configurations still resist effective analytical solutions. We propose the locus intersection method (LIM) for GCS, a hybrid method based on geometric computation and numerical search that can be used to find all the solutions for a geometric constraint problem. We show that the LIM can be used to solve all the above basic configurations.
© 2004 Elsevier Ltd. All rights reserved.

Keywords: Geometric constraint solving; Parametric CAD; Spatial basic configuration; Locus intersection

1. Introduction

Geometric constraint solving (GCS) is the central topic in much of the current work of developing parametric CAD systems. It also has applications in chemical molecular modeling, linkage design, computer vision and computer aided instruction. GCS algorithms accept the declarative description of geometric diagrams or engineering drawings as the input and output a drawing procedure. There are four main approaches to GCS: the graph analysis approach [8,13,14,19,23], the rule-based approach [1,4,10,16,20,24,25], the numerical computation approach [18,21], and the symbolic computation approach [5,17,26]. In practice, combinations of these approaches are often used to obtain the best result. The work in Refs. [3,6,9,11] uses this approach.

Since practical problems from CAD are usually very large, a basic idea in GCS is to use the following approach of *divide and conquer*.

Schema DC(P)

1. Divide the problem P into several sub-problems $P \Rightarrow P_1 \cup \cdots \cup P_m$.
2. Solve the sub-problems $Q_i = \mathrm{DC}(P_i)$ recursively with algorithm DC.
3. Merge the sub-problems together $Q_1 \cup \cdots \cup Q_m \Rightarrow P$.

Most GCS methods may be understood in this way. For instance, in the numerical computation method, we treat each individual primitive as a sub-problem in the first step. The second step is trivial and the third step is to merge these primitives by solving a large-scale equation system simultaneously. This is of course an extreme case. In general, graph algorithms are used to find feasible decompositions. The ideal decomposition is that each of the three steps in the algorithm may be solved effectively, stably and completely.

The triangle merge approaches proposed in Refs. [13,23] may be understood as decomposing a problem into three sub-problems such that each pair of sub-problems shares a common geometric primitive, and the merge step is to place one primitive with respect to the other two. In this approach, the *triangle problems*, i.e. the constraint problems consisting of three primitives, play an important role. They are used as building blocks for larger scale constraint problems. We may call them the *basic configurations* in these approaches. A smallest constraint problem that cannot be solved with the above method may also be considered as a basic configuration. If we can solve this basic problem,

* Corresponding author.
E-mail addresses: cmh@cs.purdue.edu (C.M. Hoffmann); xgao@mmrc.iss.ac.cn (X.-S. Gao).

[1] Partially supported by Chinese NSF under an outstanding youth grant (No. 69725002) and by a National Key Basic Research Project of China (No. G1998030600).

[2] Partially supported by NSF grants EIA 02-16131, DMS/CCR 01-38098 and by ARO contract 39136-MA.

then we may use it as a building block to solve an enlarged class of constraint problems. In Ref. [7], such a basic configuration with six primitives is solved with four-bar linkages. In Ref. [20], two rigid structures connected by three constraints are considered as basic configurations. In Ref. [14], a general method to find solvable sub-problems is proposed. The advantage of this kind of generic solver is that once we know how to solve a basic configuration and how to decompose a large problem into sub-problems based on this basic configuration, we may create templates to solve classes of problems. Thus, highly stable and efficient solvers become possible.

In the spatial case, two kinds of basic configurations among points and planes, tetrahedra and octahedra, are identified and studied in Refs. [3,13]. Several octahedral problems involving points and lines are also considered in Refs. [3,15]. In this paper, we will give a systematic study of basic configurations involving points, planes, and lines.

Adding lines as new primitives increase the difficulty in finding the basic configurations and solving them completely. This may be understood in two ways. First the constraint problems with lines are more difficult to solve [15]. Second, the structures for basic configurations become more complicated and the number of basic configurations increases drastically. If considering points and planes only, the smallest non-sequential basic configurations contain six primitives [14]. When adding lines as a new primitive, we show that there exist 1, 17, and 683 basic non-sequential configurations with 4, 5, and 6 primitives—even if we treat points and planes as the same type of primitives. This makes it impossible to find all these configurations manually. In this paper, an automated method is proposed to find these basic configurations.

Since GCS is often used in the conceptual design, it would be desirable to find all the solutions of the problem. This makes it difficult to use iteration methods such as the Newton–Raphson method. Homotopy methods are capable of finding all the solutions. But a pre-condition for efficient computation with this method is to find a reasonably small bound for the number of solutions of the system. For the octahedral configurations involving points and planes, efficient homotopy methods are developed by reducing the equation system for the configurations to a core system of three equations [3], and in Ref. [22] a system of two quartic equations is derived. But for basic configurations involving lines, the solution bounds are still too high to solve the problem efficiently with the homotopy method [15].

On the other hand, in Ref. [7] general linkages are used to solve 2D basic configurations efficiently. In this approach, loci of certain points in the linkage are generated and the constraint problems are solved by finding the intersections of these loci. This approach is also used to solve one basic octahedral configuration with lines in Ref. [15]. The idea of locus intersection is used to solve 2D constraint problems in Refs. [8,9]. The geometric primitives considered in Refs. [8,9] have two degrees of freedom, so Hsu and Bruderlin only remove one constraint in each step in order to generate a locus. They later considered higher-dimensional situations and apply a multivariate secant method to find solutions to the (higher-dimensional) loci. However, their approach then was not very robust in higher dimensions, and could not guarantee that they would find all solutions.

In this paper, the idea of locus intersection is extended and formalized to a general method for GCS: *locus intersection method* (LIM). The LIM is a hybrid method, which combines geometric construction and heuristic search. Theoretically, it can be used to find all the solutions for geometric constraint problems. Practically, it is powerful enough to solve many difficult problems. In fact, all the 701 basic configurations, with 4, 5 and 6 primitives, can be solved with the LIM.

The rest of this paper is organized as follows. In Section 2, we show how to find the basic configurations containing up to six primitives. In Section 3, we introduce the LIM. In Section 4, we show how to solve the basic configurations with the LIM. In Section 5, we present our conclusions.

2. Automated generation of basic configurations

We consider three types of *geometric primitives*: points, planes, and lines and two basic types of constraints: the angle constraints between line/line, line/plane, plane/plane and the distance constraints between point/point, point/line, point/plane and line/line. Angle and distance constraints between two primitives o_1 and o_2 are denoted by $\mathrm{ANG}(o_1, o_2)$ and $\mathrm{DIS}(o_1, o_2)$, respectively. If we do not need to distinguish the types of constraints, we use $\mathrm{CONS}(o_1, o_2)$ to denote the constraint between the two primitives. Parallel and incidence constraints may be considered special cases of angle and distance constraints. Between a pair of lines, there might exist both an angle constraint and a distance constraint.

We may 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 primitive o in a constraint problem, let $\mathrm{DOF}(o)$ be the degree of freedom for o and $\mathrm{DEG}(o)$ the number of constraints involving o. Note that each parallel constraint between line/line, line/plane, plane/plane will be counted twice, because it consumes two degrees of freedom for the corresponding primitives. For a set of geometric primitives O in a constraint problem, let $\mathrm{DOF}(O) = \sum_{o \in O} \mathrm{DOF}(o)$ and $\mathrm{DEG}(O)$ the number of constraints among primitives in O.

Basic configuration. A geometric constraint problem P is called a *basic configuration* if it satisfies the following conditions.

1. It is a structurally well-constrained problem [3]. Intuitively, this means that every sub-problem Q of P satisfies $\mathrm{DOF}(Q) = \mathrm{DEG}(Q) + 6$. We subtract six here,

because a rigid generic configuration in space has six degrees of freedom.

2. There is no geometric primitive o in the problem satisfying $\mathrm{DEG}(o) \le \mathrm{DOF}(o)$. That is, no primitive in the problem can be solved by a sequential construction. Details for this case can be found in Section 3.1.
3. P cannot be decomposed into smaller basic configurations. This criterion depends on what decomposition method is used. In this paper, we use the method proposed in Ref. [13].

A basic configuration with n points, m planes, and k lines is said to be of type $n\mathrm{P}m\mathrm{H}k\mathrm{L}$. Since the degrees of freedom for points and planes are both three, we sometimes do not distinguish them in the structural analysis. A basic configuration with u points and planes and v lines is denoted by $u\mathrm{B}v\mathrm{L}$. For convenience, zero is always omitted from the above notation. For instance, 0P1H5L is simplified to 1H5L.

In Ref. [13], it is pointed out that the octahedral problem in Fig. 1 is a smallest basic configuration if the geometric primitives are points and planes. There are eight non-trivial octahedral problems involving points and planes [3]. Octahedral problems 5B1L, 4B2L, and 3B3L are also considered in Ref. [3]. It is pointed out in Ref. [3] that octahedral configurations with more lines than points are always underconstrained. This is because, only angle constraints between two lines are considered. If we consider both the distance and angle constraints between two lines, there exist many more basic configurations with six geometric primitives. In the following sections, we will give all the basic configurations involving up to six geometric primitives.

2.1. Basic configurations involving four geometric primitives

Let p be a point or a plane and o a geometric primitive. Then there exists at most one constraint between p and o.

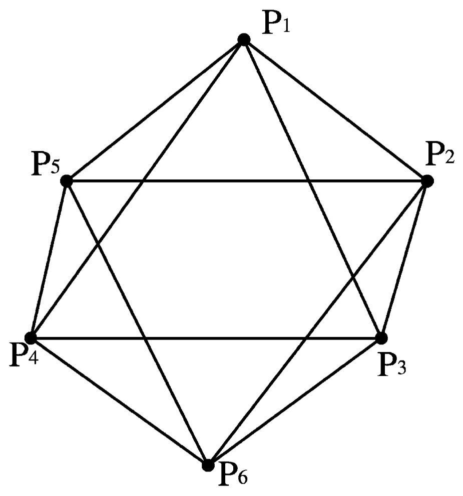

Fig. 1. Octahedron: a smallest basic configuration for points and planes.

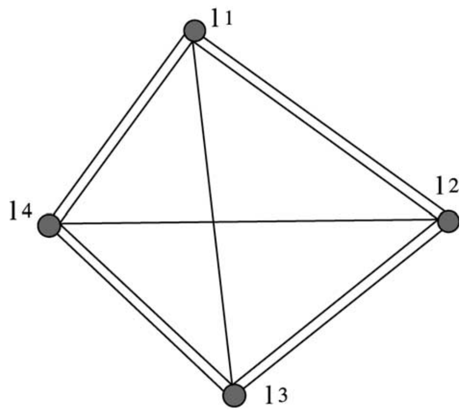

Fig. 2. The basic configuration with four lines.

If there is a point or a plane p in the constraint problem and p has less than four constraints, i.e. $\mathrm{DEG}(p) \le 3$, then p can be constructed sequentially and the problem is not a basic configuration. Therefore, all the primitives in a basic configuration with four elements must be lines. It is not difficult to check that the only basic configuration is the tetrahedron in Fig. 2. The solution to this problem can be found in Section 4.1.

Note that at least one of the constraints between l_1, l_3 and l_2, l_4 must be a distance constraint. Otherwise, there would be conflicts among the angle constraints. Hence, there exist two types of basic configurations involving four lines: $4\mathrm{L}_1$ and $4\mathrm{L}_2$ which have four and five angle constraints, respectively.

In Table 1, a means the existence of angle constraints between two lines, d means the existence of distance constraints between two lines, and a/d means the existence of both kinds of constraints.

2.2. Automated generation of basic configurations

We use the adjacency matrix to specify graphs of different constraint problems that have the same group of primitives.

When the number of geometric primitives increases, the number of constraint problems increases dramatically. Therefore, we need to automate the process of finding basic configurations. There are four basic steps to construct the basic configurations with a fixed number of primitives.

S1 Construct all possible well-constrained problems.
S2 Delete those that can be constructed sequentially with algorithm LIM0 in Section 3.1.

Table 1
Basic configurations with four lines

Pair of lines	l_1, l_2	l_2, l_3	l_3, l_4	l_4, l_1	l_2, l_4	l_1, l_3
Problem $4\mathrm{L}_1$	a/d	a/d	a/d	a/d	d	d
Problem $4\mathrm{L}_2$	a/d	a/d	a/d	a/d	d	a

S3 For those problems that have the same structure, we need to consider only one of them. Two constraint problems are said to have the same *structure* if by renaming the primitives of the same type in one of them, they become the same constraint problem.

S4 Delete those that can be decomposed into smaller problems with the method in Ref. [13].

Steps S1 and S3 can be done in a naive manner by considering all the possible combinations. We introduce several search strategies to enhance the search efficiency. For S1, the main idea is to start from a good initial constraint problem. From the definition of basic configuration, it is easy to see that each geometric primitive in the problem must be constrained with at least four constraints. For instance, if we consider problems with six vertices, the octahedron in Fig. 1 could be used as a starting point. Although it might not cover all the conditions as we have shown in Fig. 9, which is not of the octahedron type, it does enhance the search efficiency.

For Step S3, we may use the Hamming number to reduce the number of comparisons among the adjacency matrix entries. Let $A(n,n)$ be the *adjacency matrix* for a constraint problem with n primitives. $A(i,j)$ is the number of constraints between the ith primitive and the jth primitive. The Hamming number for the ith primitive is

$$H_i = \sum_{j=1}^{n} A(i,j).$$

Then a necessary condition for two adjacency matrix problems to be structurally the same is that the Hamming numbers for the points, planes, and lines in them must also be the same, respectively. Furthermore, if we cannot distinguish two problems by comparing their Hamming numbers, a test to find equivalent problems has to consider only different combinations for those primitives with the same class and the same Hamming number. For instance, let $\{3,4,4,5\}$ be the Hamming numbers for points p_1,p_2,p_3,p_4 and q_1,q_2,q_3,q_4 in two problems, respectively. Point p_1 with Hamming number 3 can only be matched with point q_1 with the same Hamming number. We need to compare only the corresponding arrays from the adjacency matrix of p_1,p_2,p_3,p_4 with that from the adjacency matrix of q_1,q_2,q_3,q_4.

2.3. Basic configurations involving five and six geometric primitives

Two simplifications are made in the process of finding basic configurations. First, we consider only the structure of the configurations and ignore the types of constraints between two lines. Second, we will treat points and planes as the same type of geometric primitives. This is possible since they have the same degree of freedom. Following the method in Section 2.2, we find all the basic configurations with five and six geometric primitives.

In Tables 4 and 5, N_0 is the number of basic configurations; N_1 is the number of problems that can be solved with method LIM1 (see Section 3.2 for definition); N_2 is the number of problems that can be solved with method LIM2; N_t is the number of basic configurations if we treat points and planes differently. We do not determine the number N_t in all cases so as to avoid the degenerated cases, which will be discussed in later work.

3. Locus intersection method for GCS

It is desirable to find all the solutions of the constraint problem. One way to do that is to find a geometric or analytical solution to a geometric constraint problem. But many difficult problems, including many basic configurations discussed in this paper, still resist such solutions [15]. In this section, we introduce a method which is capable of finding all solutions of geometric constraint problems.

3.1. Geometric construction sequence

Before presenting the method, we first define the concept of construction sequences (CSs). As we mentioned before, a primitive o can be solved sequentially if it satisfies the condition $\mathrm{DEG}(o) \le \mathrm{DOF}(o)$. Let $o_1,\ldots,o_k$ be the geometric primitives that have constraints with o (if o_i has two constraints with o, o_i will appear twice). Then o can be constructed from $o_1,\ldots,o_k$ by a construction

$$o = \mathrm{INTER}(o_1,\ldots,o_k). \tag{1}$$

For instance, let o be a point, o_1,o_2,o_3 three planes, where the distances between o and o_1,o_2,o_3 are zero. Then o is the intersection of the three planes.

In Eq. (1), if $\mathrm{DOF}(o)=k$, the corresponding construction is called *well-defined* [13]. A well-defined construction c generally introduces a finite number of solutions, which is denoted by $\mathrm{NUM}(c)$. If $k < \mathrm{DOF}(o)$, the corresponding construction is called *deficient*. In this case, o generally has an infinite number of solutions. For instance, if o is a line and o_1 and o_2 are two planes imposing angle constraints on the line, then o is a line whose direction is determined by o_1 and o_2, but can translate freely in the space.

Considering three kinds of primitives: points, planes, and lines, there are in all 10 well-defined constructions for points, six well-defined constructions for planes, and 23 well-defined constructions for lines. Here are some examples of these constructions, where P_i, H_i, L_i represent points, planes, and lines, respectively.

- $P = \mathrm{INTER}(P_1, H_1, L_1)$. The sets of points P satisfying the constraints $\mathrm{DIS}(PP_1) = d_1$, $\mathrm{DIS}(PH_1) = d_2$, $\mathrm{DIS}(PL_1) = d_2$ are a sphere, two planes, and a cylinder, respectively, and P is the intersection of them.
- $H = \mathrm{INTER}(P_1, P_2, P_3)$. Constraint $\mathrm{DIS}(H, P_i) = d_i$ implies that plane H is tangent to the sphere with P_i as center and with d_i as the radius. Then H is the common tangent plane of three spheres.
- $L = \mathrm{INTER}(P_1, P_2, P_3, L_1)$, where the constraint between L and L_1 is a distance constraint. Constraint $\mathrm{DIS}(L, P_i) = d_i$ implies that line L is tangent to the sphere with P_i as center and with d_i as the radius. Constraint $\mathrm{DIS}(L, L_1) = d_4$ implies that line L is tangent to the cylinder with L_1 as the axis and with d_4 as the radius. Then L is the common tangent line of three spheres and a cylinder.

Of these constructions, the ones introducing points and planes are relatively easy to compute. We may divide a well-defined construction introducing a line into one of three classes: 2A2D, 1A3D, or 4D, where nAmD means that in the construction there are n angle constraints and m distance constraints, respectively. There cannot be more than two angle constraints in such a construction. Here is a description of the three constructions.

2A2D
Of the three construction types for lines, this is the only one that is easy to compute. We may first determine the direction of the line from the two angle constraints, and then use the other two distance constraints to position the line. This construction generally has eight solutions.
1A3D
This class has many cases since the angle constraint may be on a line or a plane and the distance constraints may be on points or lines.
4D
This class also has many cases. The case of using four points to determine a line is considered in Refs. [3,15] and has 24 solutions. Note that we consider oriented lines in this count; the number of geometric solutions is 12. The case of using four lines to determine a line has 8 solutions in the general case.

The 1A3D and 4D construction classes are too difficult to be computed dynamically. We will discuss how to treat this Problem in Section 4.4.

A diagram can be drawn sequentially if the geometric primitives in the diagram can be listed in an order

$$o_1, o_2, \ldots, o_m \tag{2}$$

such that each o_i is introduced by a construction using primitives $o_1, \ldots, o_{i-1}$. Let C_i be the construction introducing primitive o_i. We say that the diagram can be drawn with the following *construction sequence* (abbr. CS)

$$C_1, C_2, \ldots, C_m. \tag{3}$$

Suppose that all the primitives in Fig. 1 are points. If we remove constraint $\mathrm{CONS}(p_2, p_5)$, a CS is as follows.

$$\begin{aligned} &p_1 \text{is a free point} \\ &p_2 = \mathrm{INTER}(p_1) \\ &p_3 = \mathrm{INTER}(p_1, p_2) \\ &p_6 = \mathrm{INTER}(p_2, p_3) \\ &p_4 = \mathrm{INTER}(p_1, p_3, p_6) \\ &p_5 = \mathrm{INTER}(p_1, p_4, p_6). \end{aligned} \tag{4}$$

The following algorithm to find a CS for a constraint problem is well-known (see, e.g. Ref. [6]). Since we will use it many times in this paper, we restate it here.

Algorithm LIM0.

Input: a constraint problem.
Output: a construction sequence CS with initial value $\emptyset$.

1. Let o be a primitive in the problem satisfying $d = \mathrm{DEG}(o) \leq \mathrm{DOF}(o)$.
2. Let $o_1, \ldots, o_d$ be the primitives having constraints with o. Then add a construction $o = \mathrm{INTER}(o_1, \ldots, o_d)$ to CS. Terminate if no such o exists.
3. Remove o and the constraints involving o from the problem and go to Step 1.

This algorithm is linear in the number of constraints and primitives if implemented properly.

To compute a CS, we need to introduce several concepts.

Since a spatial rigid body has six degrees of freedom, some primitives in the problem must be placed with respect to a coordinate system to guarantee that the problem has a finite number of solutions. If a constraint problem may be described by a CS, we may find a set of *placement primitives* as follows. Let d be the maximal number such that the first d primitives O_d in the CS form a well-constrained problem with the constraints in the CS. That is, O_d satisfies $\mathrm{DOF}(Od) = \mathrm{DEG}(Od) + 6$. Then these first d primitives may be treated as a set of placement primitives. In CS (4), $\mathrm{PS} = \{p_1, p_2, p_3\}$ is a set of placement primitives that form a rigid body.

The *driving primitives* in a CS are those primitives introduced by deficient constructions, but are not placement primitives. To evaluate the CS, we need to add new constraints depending on free parameters to the deficient constructions to change them to well-defined constraints. For instance, the construction $p_6 = \mathrm{INTER}(p_2, p_3)$ in Eq. (4) is a deficient construction. Point p_6 is on the intersection circle of two spheres. Then, we may use the angle formed by planes $h = p_6p_2p_3$ and $p_1p_2p_3$ as a free parameter, and p_6 can now be introduced by a well-defined construction as $p_6 = \mathrm{INTER}(p_2, p_3, h)$. The number of solutions of this new construction is defined as the number of solutions of the deficient construction.

The *maximal number of branches* of a CS (3) with p placement primitives is defined as follows

$$\mathrm{NUM}_{(3)} = \mathrm{NUM}_b \prod_{i=p+1,m} \mathrm{NUM}_{(C_i)} \tag{5}$$

where NUM_b is the number of solutions for the placement part. For CS (4), the placement part is a triangle and hence $\mathrm{NUM}_b = 1$. The number of solution branches for the remaining three constructions are two. Then $\mathrm{NUM}_{(4)} = 8$.

If all the constructions are well-defined, then the evaluation for the CS is straightforward, and the corresponding constraint problem has at most $\mathrm{NUM}_{(c)}$ solutions. Otherwise, the problem generally has an infinite number of solutions.

3.2. The locus intersection method

The LIM for GCS has the following basic steps. We will use the octahedral problem in Fig. 1 as a working example. Suppose that all the primitives in this problem are points.

Algorithm LIMd. In the algorithm, the input is a well-constrained problem. The output is to find all the solutions of the problem. Here d is the number of constraints to be removed from the original problem.

1. Find A Sequential Solution. Try to solve the problem with algorithm LIM0. If it cannot be solved, go to the next step.

2. *Remove Constraints and Determine Construction Sequence.* With Algorithm REMOVE, we may find a number $d > 0$ and d constraints, where d is the smallest number such that when we remove the d constraints from the problem the remaining problem can be solved with Algorithm LIM0. Let $\mathscr{C}$ be the CS found with LIM0 after the d constraints are removed. The geometric primitives in the removed constraints are called *locus primitives*. For the problem in Fig. 1, if we remove the constraint between p_2 and p_5, algorithm LIM0 generates CS (4).

3. *Find Placement Primitives.* We use the CS $\mathscr{C}$ obtained in the preceding step to find the placement primitives. As mentioned in Section 3.1, placement elements for CS (4) are p_1, p_2, p_3.

4. *Find driving primitives.* Since d constraints are removed from the well-constrained problem, there must be some deficient constructions in $\mathscr{C}$. In CS (4), the driving primitive is p_6.

5. *Generate loci.* We allow the driving primitives to move with small steps, which will be explained below. For each position of the driving primitives, we may compute the coordinates of all primitives in the problem according to the CS. Repeat the above process, we generate a set of coordinates for each locus primitive. The set thus generated is called the *locus for the locus primitive*. So the locus here is discrete. The continuous curve in Fig. 3 is obtained by connecting two consecutive points with a line segment.

How to select the moving steps for the driving primitive is a subtle matter. The smaller the steps, the better the chance

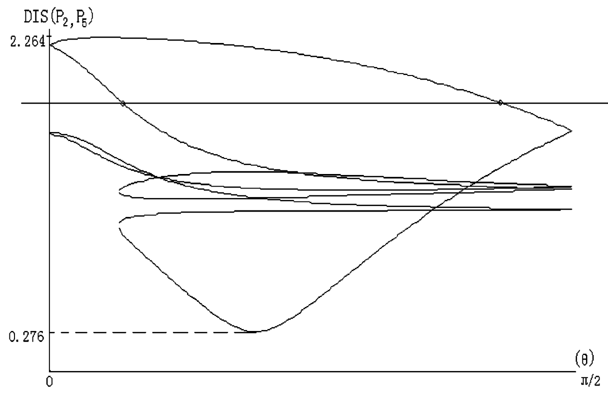

Fig. 3. The distance curves for p_2p_5 in the Octahedral problem for points. The x-axis is the driving parameter for point p_6. The y-axis is $\mathrm{DIS}(p_2, p_5)$. The horizontal line is at 1.82. The average time to compute the curves is 0.07 s on a PC with PIII733 and 128M memory. We select 157 samples for the driving primitive. Sixteen curve segments are computed, which contain 2512 points. In the figure, two consecutive points are connected with a line segment.

we find all the solutions. But, a very small step will lead to a large number of samples and will decrease the computation speed. We use the following heuristic to select a suitable set of values for the driving parameters. First, fixed and uniform steps for the driving parameters are selected and used to generate the loci. For instance, if the range for an angle parameter is from 0 to π, we would select $\pi/300$ as the initial step. Second, if for two consecutive values t_1, t_2 of the driving parameter, the distances between the positions of any locus primitive are larger than a small number, say 0.01, we will add more samples between t_1 and t_2.

In this step, we need to decide how to move the driving points. A general method is as followed. Suppose that $o = \mathrm{INTER}(o_1, \ldots, o_m)$ is a driving point with s deficient degrees of freedom. Then we may select s primitives $b_1, \ldots, b_s$, which have been constructed before o, and compute o using the following construction $o = \mathrm{INTER}(o_1, \ldots, o_m, b_1, \ldots, b_s)$. The constraints between o and b_i are called *driving constraints or driving parameters.*

We also need to determine the range from which the driving parameters take values. If the constraint between o and b_i is an angle constraint, then we may take the range for the parameter as $0 \sim \pi$ and $0 \sim 2\pi$ depending on whether o is a point or a line (or plane). If the constraint between o and b_i is a distance constraint, we may use the following result to determine an upper bound for the distance from o to b_i.

Let P be a point and o_1, o_2 any primitives. Then

$$\mathrm{DIS}(o_1, o_2) \leq (o_1, P) + \mathrm{DIS}(P, o_2)$$

The proof for this result is omitted. In the general case, such explicit upper bounds may not exist.

Note that the lower and upper bounds mentioned above are generally not optimal (see Figs. 4 and 5). A more detailed analysis could reveal narrower enclosures for feasibility ranges. For example, Ref. [12] derives tight

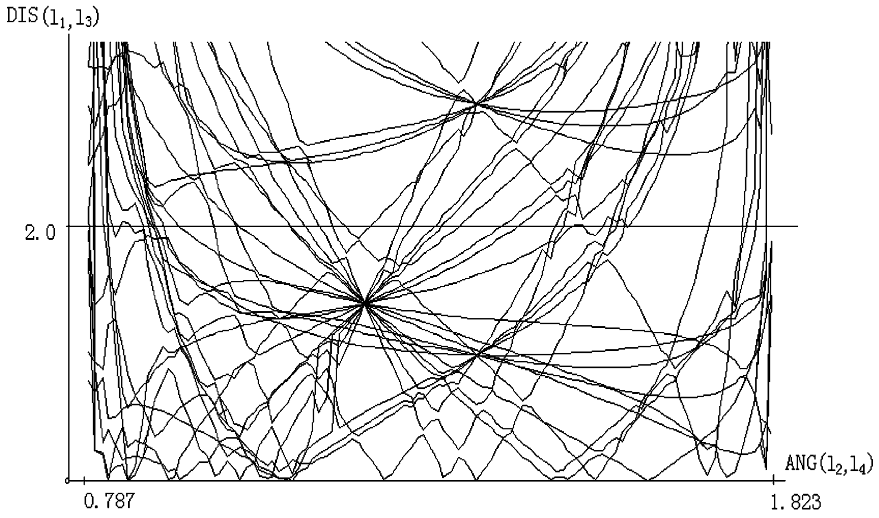

Fig. 4. Solutions to configuration 4L$_1$. The x-axis is ANG(l_2, l_4). The y-axis is the DIS(l_1, l_3). The average time to compute the 64 curve segments is 0.23 s on a PC with PIII733 and 128M memory. We select 314 samples for the driving primitive. Sixty four curve segments are computed, which contain approximately 6656 points.

range estimates for valid parameter ranges in 2D GCS based on geometric reasoning.

In Eq. (4), we use the angle between plane $p_6p_2p_3$ and plane $p_1p_2p_3$ as the driving parameter. This parameter may change from 0 to π, but we consider only values from 0 to $\pi/2$ by symmetry.

6. *Locus intersection*. After the loci for locus primitives are generated, we check them to find whether the removed constraints are satisfied. To be more precise, let DS be the set of values for the driving parameters, LS the set of locus primitives, and PS the set of all primitives. For each value t in DS, we may compute the coordinates LS(t) and PS(t) for the primitives in LS and PS. We search the set LS(t) to find the set T of values t_0 such that LS(t_0) satisfies the removed constraints approximately. For instance, let the removed constraint be DIS(o_1, o_2). For each value t in DS, compute $d(t) = |\text{DIS}(o_1, o_2) - |o_1(t)o_2(t)||$. A value t_0 is a solution if the following conditions are satisfied: $d(t_0)$ is a local minimum and $d(t_0)$ is a small number. Then the solutions of the GCS problem are $\{\text{PS}(t) | t \in T\}$. This step may be considered to find the intersections of several loci.

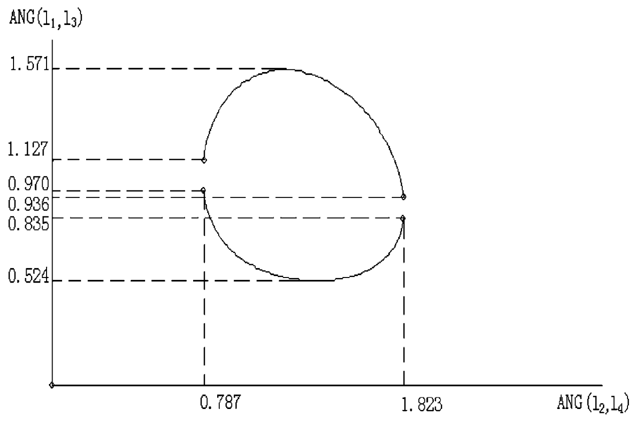

Fig. 5. Solutions to configuration 4L$_2$. The x-axis is ANG(l_2, l_4). The y-axis is the ANG(l_1, l_3). The average time to compute the 64 curves is 0.23 s on a PC with PIII733 and 128M memory. The curves contain approximately 6656 points.

Fig. 3 is the distance curve for the 6p problem with a set of constraint values given in Table 2. From this figure we may give a classification of number of solutions for the problem (in Table 3) according to distance p_2p_5.

For the problem given in Table 2, $|p_2p_5| = 1.3$. There exist 12 solutions. All of them are realizable in Euclidean space.

Algorithm REMOVE. The input is a well-constrained problem. The output is an integer $d \geq 0$ and a set $\mathcal{S}$ of d constraints. d is the smallest number such that when we remove the d constraints in $\mathcal{S}$ from the problem the remaining problem can be solved with Algorithm LIM0.

1. We will use a list $\mathcal{L}$ and some pointers L_i which will be pointed to notes of $\mathcal{L}$.
2. For a vertex v, let SDEG(v) = DEG(v) − DOF(v). For each vertex v, compute e = SDEG(v). We put v into list $\mathcal{L}$ in such a way that L_e is pointed to the first vertex w with SDEG(w) = e and the vertices are in an increasing order in $\mathcal{L}$ with SDEG(v). These properties for $\mathcal{L}$ will be kept during the algorithm. Let $d = 0$, $\mathcal{S} = \emptyset$.
3. Let v be the first vertex in $\mathcal{L}$ and m = SDEG(v).
4. If $m = 0$, then v can be constructed explicitly. Go to Step 7.

Table 2
A set of constraint values for the 6p problem

$d_{12} = 1.000$	$d_{13} = 1.108$	$d_{23} = 1.073$	$d_{14} = 1.181$
$d_{15} = 1.275$	$d_{26} = 1.171$	$d_{34} = 1.157$	$d_{45} = 1.164$
$d_{46} = 1.067$	$d_{56} = 1.110$	$d_{36} = 1.052$	$d_{25} = 1.3$

Table 3
Number of solutions of the Octahedron problem for different values of p_2p_5

Values for $\|P_2P_5\|$	0.275–1.171	1.171–1.228	1.228–1.353	1.353–1.615	1.615–2.263
Number of solutions	4	8	12	8	4

5. If $m > 0$, then v cannot be constructed explicitly. Set $d = d + m$. Remove m edges (constraints) connecting v and add these edges to $\mathcal{S}$.
6. For each vertex w connected with v with these m edges, subtract the number c of removed edges between w and v from DEG(w). Since DEG(w) is changed, we move w to the correct position in $\mathcal{L}$. Now, v can be constructed explicitly. Go to Step 7.
7. Remove vertex v from $\mathcal{L}$. For each vertex w connected with v, subtract the number c of edges between w and v from DEG(w) and move w to the correct position in $\mathcal{L}$.

Let n and e be the number of vertices and edges in the constraint graph. Steps 3–7 are the main loop in the algorithm. In each loop, we will remove one vertex from the graph. Therefore, the loop runs n times. Step 2 needs $O(n+e)$ operations. Let v_i be the vertex removed in the ith loop. Steps 3 and 4 need O(1) operations. Steps 5, 6, and 7 need $O(\mathrm{DEG}(v_i))$ operations. Therefore, the total operations for steps 3–7 are $O(n) + O(\sum_{i=1}^{n} \times \mathrm{DEG}(v_i)) = O(n) + O(e) = O(n+e)$. Thus, the complexity of the algorithm is $O(n+e)$.

Let d be the number of constraints cut. Then the driving parameters take values in a d-dimensional space. Note that number d is a critical parameter influencing the amount of computation required by the above method. Because of this, we call the above method the *locus intersection method of dimension d* (abbr. LIMd).

Another important parameter in the above method is the number of branches of the CS which represents the number of loci need to be computed. This bound is generally much smaller than the Bezout number (see, e.g. Ref. [3]) of the problem. For instance, the naive Bezout number for the Octahedron problem in Fig. 1 is 4096. In Ref. [3], a system of polynomial equations with total degree 64 and BKK bound 16 is obtained after simplification. The number of loci need to be computed in our case is 8. Note, however, that the loci here are different from the paths used in the homotopy method. One locus here may give rise to multiple solutions of the problem, and therefore the number of loci is not equal to the number of solutions.

Table 4
Basic configurations with five geometric primitives

	5B	4B1L	3B2L	2B3L	1B4L	5L	Sum
N_0	0	0	1	1	3	12	17
N_1	0	0	1	1	3	12	17
N_t	0	0	1	1	3	12	17

4. Solving the basic configurations

We have used the LIM method in Section 3.2 to solve all basic configurations obtained in Section 2.2. The two tetrahedral problems for lines can be solved with algorithm LIM1. The results for the basic configurations with five and six primitives are given in Tables 4 and 5. It is readily seen that *all the basic configurations up to six vertices for points, planes, and lines can be solved with method LIM1 and LIM2*. Below, we will give some of the details of the solutions.

4.1. Basic configurations with four lines

We first consider the basic configurations with four lines (Fig. 2). If we remove the constraint between l_1 and l_3, a CS is as follows.

$$\begin{aligned} &l_1 \text{ is a free line} \\ &l_2 = \mathrm{INTER}(l_1, l_1) \\ &l_4 = \mathrm{INTER}(l_1, l_1, l_2) \\ &l_3 = \mathrm{INTER}(l_2, l_2, l_4, l_4). \end{aligned} \tag{6}$$

The placement primitives are lines l_1 and l_2. The driving primitive is l_4 which has an angle and distance constraint with l_1. Since one of the constraints CONS(l_1, l_3) and CONS(l_2, l_4) is a distance constraint, we may always assume that the constraint CONS(l_2, l_4) is a distance constraint and add an angle constraint between l_2 and l_4 as the driving constraint.

Figs. 4 and 5 give the solution curves for problems $4L_1$ and $4L_2$ with concrete constraint values given in Tables 6 and 7.

Note that the Bezout numbers for problems $4L_1$ and $4L_2$ are 4,194,304 and 2,097,152. The maximal number of branches (see Eq. (5) for the definition) for Eq. (6) is 64 in both cases. Therefore, the number of loci generated by the LIM1 method is 64. The two figures look different, because the curve in Fig. 5 is the overlap of many curve segments. For the given values of constraints, we may obtain

Table 5
Basic configurations with six geometric primitives

	6B	5B1L	4B2L	3B3L	2B4L	1B5L	6L	Sum
N_0	1	1	4	12	39	132	494	683
N_1	1	1	4	12	37	122	437	614
N_2	0	0	0	0	2	10	57	69
N_t	8	9	?	?	?	?	?	?

Table 6
The set of constraints for $4L_1$

$d_{12} = 0.00000$	$a_{12} = 0.78540$	$d_{23} = \sqrt{2}$	$d_{24} = 1.5$
d_{13} removed	$d_{14} = 1.00000$	$a_{14} = \pi/2$	$d_{34} = 2$
$a_{34} = \pi/3$	$a_{23} = \pi/4$		

Table 7
The set of constraints for $4L_2$

$d_{12} = 0.00000$	$a_{12} = 0.78540$	$d_{23} = \sqrt{2}$	$d_{24} = 1.5$
a_{13} removed	$d_{14} = 1.00000$	$a_{14} = \pi/2$	$d_{34} = 2$
$a_{34} = \pi/3$	$a_{23} = \pi/4$		

the following results from the method.

1. Problem $4L_1$ has solutions for $0.787 < \mathrm{ANG}(l_2, l_4) < 1.823$. Problem $4L_2$ has solutions for $0.787 < \mathrm{ANG}(l_2, l_4) < 1.823$ and $0.524 < \mathrm{ANG}(l_1, l_3) < 1.571$. Note that the curves in Figs. 4 and 5 are multiple curves, i.e. several curves are coincident.
2. Problem $4L_2$ has 64 solutions on the solid part of the solution curve. Problem $4L_1$ has at most 134 solutions for different values $\mathrm{DIS}(l_1, l_3)$. Fig. 4 shows the case for $\mathrm{DIS}(l_1, l_3) = 2.0$, which has 98 solutions.

The basic configuration $4L_2$ can be solved analytically. We may draw l_1 and l_2 first and determine the directions of l_3 and l_4 by the angle constraints. Now, l_3 and l_4 can be determined by solving a set of four equations of degree two. In this way, we may prove that $4L_2$ has at most 64 solutions.

4.2. Basic configurations with five primitives

In all, there are 17 basic configurations involving five primitives if we treat points and planes as the same type. All of these problems can be solved with method LIM1. We use a 5L configuration in Fig. 6 as an illustrative example, where $\mathrm{CONS}(l_1, l_3)$, $\mathrm{CONS}(l_1, l_4)$, and $\mathrm{CONS}(l_2, l_4)$ are distance constraints and $\mathrm{CONS}(l_2, l_5)$ is an angle constraint.

For this 5L configuration, each of the 14 constraints in the configuration can be removed and we may obtain 14 CSs for the problem. Among these CSs, the following two are essentially different. All other CSs are similar to one of them.

If we remove $\mathrm{CONS}(l_1, l_3)$, a CS is as followed

$$\begin{aligned} &l_1 \text{ is a free line} \\ &l_5 = \mathrm{INTER}(l_1, l_1) \\ &l_4 = \mathrm{INTER}(l_1, l_5, l_5) \\ &l_2 = \mathrm{INTER}(l_1, l_1, l_4, l_5) \\ &l_3 = \mathrm{INTER}(l_2, l_2, l_4, l_4) \end{aligned} \tag{7}$$

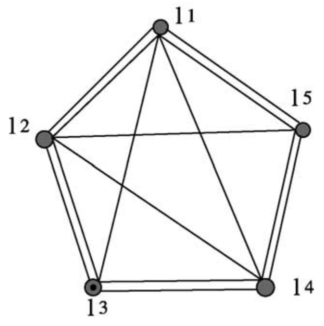

Fig. 6. One 5L basic configuration.

The placement primitives are l_1, l_5. The driving primitive is l_4. The maximal number of branches is 512.

If we remove $\mathrm{CONS}(l_2, l_5)$, a CS is as followed

$$\begin{aligned} &l_1 \text{ is a free line} \\ &l_2 = \mathrm{INTER}(l_1, l_1) \\ &l_3 = \mathrm{INTER}(l_1, l_2, l_2) \\ &l_4 = \mathrm{INTER}(l_1, l_2, l_3, l_3) \\ &l_5 = \mathrm{INTER}(l_1, l_1, l_4, l_4) \end{aligned} \tag{8}$$

The placement primitives are l_1, l_2. The driving primitive is l_3.

In Step 2 of algorithm LIMd, we may remove different constraints. Therefore, the process of obtaining a CS is not unique, and we could obtain many different CSs for the same problem. An advantage of generating multiple CSs is that we may select a ‘good’ one. For instance CS (7) is better than (8). This is because, all the construction steps in Eq. (7) determine a line using two angle and two distance constraints. This kind of construction is easy to evaluate. In Eq. (8) we construct l_4 using one angle and three distance constraints. This kind of construction is difficult to evaluate. Therefore, we will choose CS (7) to compute the problem. A general heuristic to select a good CS is to have the smallest number of lines as the non-placement elements, and in constructions for lines to use more 2A2D type constructions.

4.3. Basic configurations with six primitives

From Table 5, there are in all 683 basic configurations involving six primitives if we treat points and planes as the same type. Of these problems, 614 can be solved with method LIM1 and 69 have to be solved with method LIM2. We take the 2P4L configuration in Fig. 7 as an illustrative example. To solve this problem, we need algorithm LIM2. The following construction is such that all steps are easy to

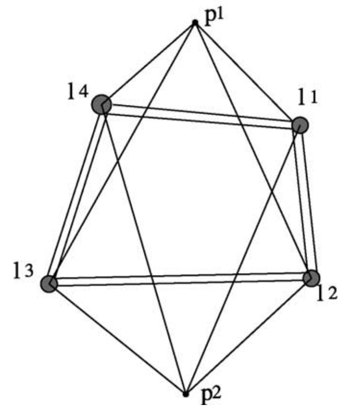

Fig. 7. One 2P4L basic configuration.

evaluate. The Construction Sequence is as follows:

$$\begin{aligned} &l_1 \text{ is a free line}\\ &l_2 = \mathrm{INTER}(l_1, l_1)\\ &l_3 = \mathrm{INTER}(l_2, l_2)\\ &l_4 = \mathrm{INTER}(l_1, l_1, l_3, l_3)\\ &p_1 = \mathrm{INTER}(l_2, l_3, l_4)\\ &p_2 = \mathrm{INTER}(l_2, l_3, l_4) \end{aligned} \tag{9}$$

The removed constraints are CONS(p_1, l_1) and CONS(p_2, l_1). The placement primitives are l_1, l_2. The driving primitive is l_3. The maximal number of branches is 4096.

Note that besides the constraints with l_2, the line l_3 still has two degrees of freedom. According to the algorithm LIM2, we need to add two more driving constraints to construct l_3. The new construction for l_3 is

$$l_3 = \mathrm{INTER}(l_2, l_2, l_1, l_1)$$

and the driving parameters are the angles and distances between l_3 and l_1. An upper bound for the DIS(l_3, l_1) is DIS(p_1, l_1) + DIS(p_1, l_3) by the fact mentioned in Step 5 of Algorithm LIMd.

Since there are two driving parameters, the loci for P_1 and P_2 are surfaces, and we need to search two surfaces in the loci intersection step. We may use the following trick to reduce the amount of computation and searches. After the computation of point P_1, we will check whether constraint CONS(P_1, l_1) is satisfied, and only use those P_1 satisfying this constraint to compute P_2. In this way, the locus for P_2 will be a curve.

We consider the 6L configuration in Fig. 8 where CONS(l_1,l_4), CONS(l_1,l_5), CONS(l_2,l_6), CONS(l_3,l_4), CONS(l_3,l_6) are distance constraints and CONS(l_2, l_5) is an angle constraint. This problem can be solved with LIM2. If we remove CONS(l_6, l_2) and CONS(l_6, l_3), a CS is as follows.

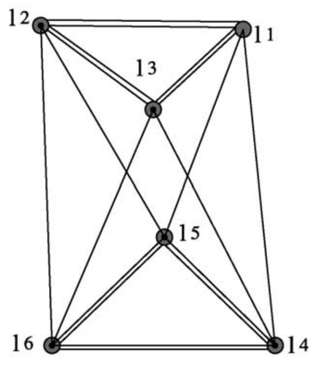

Fig. 8. One 6L basic configuration.

$$\begin{aligned} &l_1 \text{ is a free line}\\ &l_2 = \mathrm{INTER}(l_1, l_1)\\ &l_3 = \mathrm{INTER}(l_1, l_1, l_2, l_2)\\ &l_4 = \mathrm{INTER}(l_1, l_3)\\ &l_5 = \mathrm{INTER}(l_1, l_2, l_4, l_4)\\ &l_6 = \mathrm{INTER}(l_4, l_4, l_5, l_5) \end{aligned} \tag{10}$$

The placement primitives are l_1, l_2, l_3. The driving primitive is l_4.

This problem differs from all other examples in that the placement part is non-trivial. We need to estimate the number of solutions for the placement part, while the placement part for all the previous examples has only one solution.

To compute the placement part, we may use l_1, l_2 as a new placement and compute l_3. In this way, we may find eight solutions. But in these solutions, only four are different geometrically. This is because, when rotating l_1, l_2, l_3 around the common perpendicular line of l_1 and l_2 by an angle π, one solution will change into another solution. Therefore, the placement part has only four solutions. This example shows that to use a large placement part may reduce the total number of branches needed to be computed.

Each of the three constructions for l_4, l_5, l_6 in Eq. (10) will produce 8 branches of curves. Therefore, the maximum number of branches for CS (10) is $4 \times 8 \times 8 \times 8 = 2048$.

All the configurations encountered so far have the octahedron structure. However, not all problems have this structure. The 1P5L configuration in Fig. 9 is not an octahedron. This problem could be solved with the LIM2 method.

4.4. Recursive locus intersection method

A precondition that the LIMd method works is that constructions can be evaluated efficiently. This is always

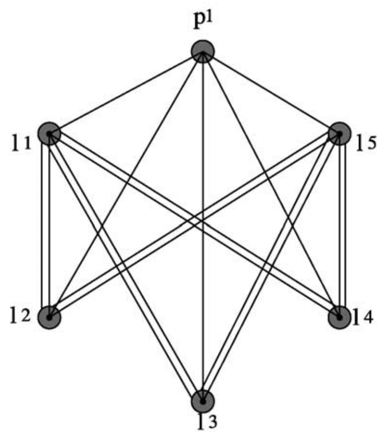

Fig. 9. A 1P5L configuration does not have an octahedron structure.

the case in basic 2D GCS, since we need to solve only two equations of degree less than or equal to two [7]. As we mentioned before, some of the constructions for spatial lines are difficult to compute. We introduce the *Recursive LIMd* method to solve this problem.

Let us consider the construction for line

$l = \text{INTER}(l_1, l_2, l_3, l_4)$

which is of class 1A3D. Suppose that $\text{CONS}(l, l_1)$ is an angle constraint. To compute this construction we introduce a new construction

$l = \text{INTER}(l_1, l_2, l_2, l_3)$

where the angle constraint between l and l_2 is used as a free parameter. In other words, we change the construction from a well-defined one to a deficient one that is easy to compute by removing another constraint $\text{DIS}(l, l_4)$. The removed constraint needs to be checked later in the intersection step. If the original problem involving this construction may be solved with an LIM_d method, then we will use an LIM_{d+1} method to solve it.

For instance, if the constraints $\text{CONS}(l_1, l_3)$, $\text{CONS}(l_1, l_4)$, $\text{CONS}(l_2, l_4)$, $\text{CONS}(l_2, l_5)$ in the 5L problem in Fig. 6 are all distance constraints, then we need a 1A3D construction in any CS. So, if we remove $\text{DIS}(l_1, l_3)$ and $\text{DIS}(l_2, l_5)$, a CS is as follows.

$$\begin{aligned}
&l_1 \text{ is a free line}\\
&l_5 = \text{INTER}(l_1, l_1)\\
&l_4 = \text{INTER}(l_1, l_5, l_5) \qquad (11)\\
&l_2 = \text{INTER}(l_1, l_1, l_4)\\
&l_3 = \text{INTER}(l_2, l_2, l_4, l_4)
\end{aligned}$$

The placement primitives are l_1, l_5. The driving primitives are l_4 and l_2. We use the angle constraints $\text{ANG}(l_4, l_1)$ and $\text{ANG}(l_4, l_2)$ as the driving parameters. In terms of efficiency, this is a trade off: we use a LIM2 method to solve a problem that can be solved with an LIM1 method to avoid the computation of a 1A3D construction.

We may treat a construction of type 4D for a line similarly. In this case, we need to remove two constraints and to introduce two driving angle constraints. If the original problem involving this construction may be solved with an LIM_d method, then this change would let us solve the problem using an LIM_{d+2} method.

5. Conclusion

One of the main difficulties of GCS lies in the fact we need methods that deliver efficient and complete solutions to the constraint problem. A basic idea of GCS is to decompose the constraint problem into smaller ones into some basic configurations. In this paper, we identify all basic spatial configurations containing up to six geometric primitives and propose the LIM to find all solutions of these basic configurations.

This paper further reveals the nature of difficulties of GCS. Even for constraint problems with up to six primitives, there exist hundreds of essentially different basic configurations and most of them are quite difficult to solve. This is basically due to the presence of lines. If only points and planes are considered, then there is only one basic structure: the octahedron as shown in Ref. [3]. This suggests that we need to find a way to avoid using lines in spatial GCS in the same way as points and planes. Also, some of the basic line configurations may have special cases in which the algebraic equations must be solved with specialized techniques.

It is interesting to note that application considerations also motivate limiting the role of lines in spatial constraint systems. Namely, in the definition of spatial constraint problems in parametric engineering design, there is an underlying mechanism to select a solution based on the orientations of the elements, in relation to each other. When editing such a design, assigning new dimensional constraint values to some of the distances or angles, it is possible that the solution so identified is not correct, [2]. Such problems can be avoided in many cases when limiting the role of lines in the constraint structure.

References

[1] Brüderlin B. Using geometric rewriting rules for solving geometric problems symbolically. Theor Comput Sci 1993;116: 291–303.

[2] Chen X, Hoffmann C. On editability of feature based design. CAD 1995;27:905–14.

[3] Durand C, Hoffmann CM. A systematic framework for solving geometric constraints analytically. J Symbol Comput 2000;30(5): 493–529.

[4] Gao XS, Chou SC. Solving geometric constraint systems I. A global propagation approach. Comput-Aided Des 1998;30(1): 47–54.
[5] Gao XS, Chou SC. Solving geometric constraint systems II. A symbolic approach and decision of Rc-constructibility. Comput-Aided Des 1998;30(2):115–22.
[6] 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. LNAI No. 2061, Berlin: Springer; 2001. p. 16–25.
[7] Gao XS, Jiang K, Zhu C-C. Geometric constraint solving with conics and linkages. Comput-Aided Des 2002;34(6):421–33.
[8] Hsu C. Graph-based approach for solving geometric constraint problems. PhD Thesis, The University of Utah, 1996.
[9] Hsu C, Brüderlin B. A hybrid constraint solver using exact and iterative geometric constructions. In: Roller D, Brunet P, editors. CAD systems development—tools and methods. Berlin: Springer; 1997.
[10] Joan-Arinyo R, Soto A. A correct rule-based geometric constraint solver. Comput Graph 1997;21(5):599–609.
[11] Joan-Arinyo R, Soto A. Combining constructive and equational geometric constraint-solving techniques. ACM Trac Graph 1999; 18(1):35–55.
[12] Joan-Arinyo R, Mata N, Soto A. A constraint-solving based approach to analyze 2D geometric problems. Proc Solid Modeling '01, Ann Arbor, MI 2001;11–17.
[13] Hoffmann C, 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.
[14] Hoffmann CM, Lomonosov A, Sitharam M. Finding solvable subsets of constraint graphs. LNCS No. 1330, Berlin: Springer; 1997. p. 163–97.
[15] Hoffmann CM, Yuan B. On spatial constraint solving approaches. In: Richter-Gebert J, Wang D, editors. Proceedings of ADG'2000, Zürich, Switzerland, September. Berlin: Springer; 2003. in press.
[16] Kramer GA. Solving geometric constraints systems: a case study in kinematics. Cambridge, MA: MIT Press; 1992.
[17] Kondo K. Algebraic method for manipulation of dimensional relationships in geometric models. Comput-Aided Des 1992;24(3): 141–7.
[18] Lamure H, Michelucci D. Solving geometric constraints by homotopy. IEEE Trans Visualization Comput Graph 1996;2(1): 28–34.
[19] Latham RS, Middleditch AE. Connectivity analysis: a tool for processing geometric constraints. Comput-Aided Des 1994;28(11): 917–28.
[20] Lee JY, Kim K. Geometric reasoning for knowledge-based design using graph representation. Comput-Aided Des 1998;28(10):831–41.
[21] Lin VC, Gossard DC, Light RA. Variational geometry in computer-aided design. Comput Graph 1981;15(3):171–7.
[22] Michelucci D. Using Cayley–Menger Determinants, undated world-wide-web document, www.emse.fr/~micheluc/MENGER/
[23] Owen J. Algebraic solution for geometry from dimensional constraints. ACM symposium, found of solid modeling, New York: ACM Press; 1991. p. 397–407.
[24] Sunde G. Specification of shape by dimension and other geometric constraints. Geometric modeling for CAD application, Amsterdam: North-Holland; 1998. p. 199–213.
[25] Verroust A, Schonek F, Roller D. Rule-oriented method for parameterized computer-aided design. Comput-Aided Des 1992; 24(10):531–40.
[26] Wu WT. Basic principles of mechanical theorem proving in geometries. Beijing: Science Press; 1984. [English Version. Berlin: Springer; 1994].

Christoph M. Hoffmann graduated from the University of Wisconsin in 1974 and is Professor of Computer Science at Purdue University. His research interests are in CAD, geometric constraint solving, visualization and applications of geometric computing. He is on the editorial boards of seven scholarly journals including CAD. The author of two monographs, Hoffmann has published numerous articles and book chapters. Recently, he worked with an interdisciplinary team to model the impact of Flight 77 into the Pentagon. See http://www.cs.purdue.edu/homes/cmh/simulation/.

Weiqiang Yang is presently a graduate research assistant in the Computer Science Department, Purdue University. His research interests include CAD, CAGD, Geometric Constraint Solving and Robotics. Weiqiang Yang received the BS degree in Mechanical Engineering in North China University of Technology, the MS degree in Mechanical Engineering and Automation in Beijing Institute of Technology and the PhD degree in Applied Mathematics in Chinese Academy of Sciences.

Xiao-Shan Gao received his PhD degree from the Chinese Academy of Sciences in 1988. Since 1996, he has been a Professor in the Institute of System Science, Chinese Academy of Sciences. He has published over ninety research papers, two monographs and edited four books or conference proceedings. His research interests include: automated reasoning, symbolic computation, intelligent CAD and CAGD. Webpage: http://www.mmrc.iss.ac.cn/~xgao.