www.cadanda.com
Computer-Aided Design AND APPLICATIONS

Solving Under-constrained Assembly Problems Incrementally Using a Kinematic Method

Yong Liu[1,2,3,4], Hai-Chuan Song[1,2,3,4] and Jun-Hai Yong[1,3,4]

[1]School of Software, Tsinghua University, Beijing 100084, P. R. China
[2]Department of Computer Science and Technology, Tsinghua University, Beijing 100084, P. R. China
[3]Key Laboratory for Information System Security, Ministry of Education of China, Beijing 100084, P. R. China
[4]Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, P. R. China

ABSTRACT

In this paper, an incremental method is proposed for solving under-constrained assembly problems under interactive environment. During an assembling process, a designer adds or removes some constraints, changes the design parameters, or maneuvers some bodies frequently. For each operation, we identify and solve the affected biconnected subgraph in the constraint graph using the cut-joint method. A strategy is provided to construct the joint reference frames from the former assembly configuration. After solving the biconnected subgraph, rigid transformation is utilized to propagate the changes until all subgraphs are visited. Examples are provided to illustrate the effectiveness of the method proposed in the paper.

Keywords: assembly constraint, assembly modeling, kinematic joint, constraint solver.

1. INTRODUCTION

Assembly design plays an important role in product design activities, for most manufactured products are assemblies of individual parts. After part design, a designer constructs assembly models that contain information about the relative positions of parts, obtained by specifying assembly mating conditions between parts in a CAD system, which will solve those assembly constraints to find out the positions and postures of parts.

A common approach to solving geometric constraint problems is to use a decomposition-recombination scheme based on graph algorithms and numerical solvers [2–5,11,14,16,18,20,23]. These decomposition schemes often rely on breaking a system into rigid (also called well-constrained) subsystems. There exist two definitions of rigidity: geometric rigidity and structural (combinatorial) rigidity. A geometric constraint system is called as geometric rigidity if the system has finite solutions. A structurally rigid subsystem is characterized by some combinatorial properties of its corresponding geometric constraint graph. For a 2D bar-and-joint geometric problem, composed of universal joints connected by fixed length bars, i.e., point–point distance constraints, Laman proved that it is geometrically rigid if and only if its geometric constraint graph is (2, 3)-sparse [10]. Tay et al. [22] and Lee et al. [12] studied a special case of 3D geometric problems, i.e., body-and-bar structure and discussed the combinatorial property of its constraint graph. However, for a general 3D geometrically rigid problem, there does not exist a nice combinatorial property to be used to identify a geometric rigid subgraph. Some special approaches to solving 3D geometric problems have been proposed. Kramer [9] utilized a degree-of-freedom (DOF) analysis to determine the solution sequentially, but his approach is applied to an open loop or closed loop that can be solved sequentially. Li et al. [13] used a max-matching algorithm of digraph to decompose an assembly problem into small well-constrained subproblems. Peng et al. [19] proposed a decomposing algorithm by repeatedly reversing the directions of the edges in a constraint graph to get a better decomposition. Also, since a subproblem has always multiple solutions and a constructive solver should be able to navigate the solution space to select appropriate ones to synthesize a solution of the total system, which normally requires the solver can enumerate all solutions of a subproblem, which is too expensive to an interactive CAD system. Kim et al. [8] presented an assembly modeling system which

Taylor & Francis

divides an assembly into several independent groups and uses numerical method to solve all equations involved in each group separately. Kim et al. [7] proposed a kinematic method to handle closed loops with under-constrained states, which first converts the geometric constraints into joint relations, cuts a joint from the closed loop, and obtains the final configuration of the closed loop by solving inverse kinematics of the open loop. Based on similar thought, Xia et al. [24] converts an assembly problem into a constraint multigraph, and designates each mating constraint a weight representing the difficulty. A minimum weight spanning tree is generated, then the residual constraints converted into equations denoted by a recursive form of generalized coordinates, will be solved simultaneously.

In a realistic interactive CAD assembly system, a designer constructs an assembly step by step from scratch. The operations include bringing into or deleting from the assembly a part or mating condition, maneuvering (moving, revolving) a part with respect to the constraints, etc. In most circumstances, the constraint graph is in an under-constrained state. All those operations could invoke necessary recomputations to reflect the changes to the constraint graph many times in a complete assembly design process, then it's necessary to reduce the expensive computing as much as possible. We note that a typical product composed of many parts can be settled part by part sequentially, while only a few closed loops are required to be solved simultaneously. To reduce unnecessary computation, we use the DOF analysis [9] geometrically to analyze the residual degrees of freedom between two parts and maintain such information for the next operations incrementally. When a change in the constraint graph occurs, the affected biconnected subgraph is solved at first using the cut-joint method [6,17,24,25], and such changes are spread out only using rigid transformations. A strategy is proposed to construct the joint reference frames from the initial positions, which make the solver generally produce a result close to the configuration of the assembly before changes occurred.

The rest of this paper is organized as follows. In Section 2, we introduce some definitions of geometric constraint graph. In Section 3, we describe the details of the entire procedure. Two practical assembly examples are illustrated in Section 4. Finally, conclusions are made in Section 5.

2. GEOMETRY REPRESENTATION AND CONSTRAINT GRAPH

2.1. Geometry Representation

In the assembly design, the mating conditions of a base part and moving part can be represented by the relationships (distance, angle, etc.) of planes, lines, and points. All those constraints can be equivalently converted to a composition of some primitive constraints. Five primitive constraints were presented in [6], called *dot-1* constraint (Φ^{d1}), *dot-2* constraint (Φ^{d2}), *spherical* constraint (Φ^{S}), *distant* constraint (Φ^{dist}), and *angle* constraint (Φ^{ang}), to convert a subset of mating conditions. To completely represent the relationships (distance, angle, tangency, superpose, etc.) of any two primitives of planes, lines and points, we introduce a new primitive constraint, called *cylindrical* constraint. Let $\mathbf{a}_i$ and $\mathbf{a}_j$ be unit vectors fixed on bodies i and j, respectively, and $\mathbf{d}_{ij}$ be a vector from a point $\mathbf{p}_i$ fixed on body i to a point $\mathbf{p}_j$ fixed on body j. All primitive constraints are defined as below.

1. *Dot-1* constraint requires $\mathbf{a}_i$ and $\mathbf{a}_j$ to be orthogonal. $\Phi^{d1}(\mathbf{a}_i, \mathbf{a}_j) = \mathbf{a}_i^T \mathbf{a}_j = 0$
2. *Dot-2* constraint requires $\mathbf{a}_i$ and $\mathbf{d}_{ij}$ to be orthogonal. $\Phi^{d2}(\mathbf{a}_i, \mathbf{d}_{ij}) = \mathbf{a}_i^T \mathbf{d}_{ij} = 0$
3. *Distant* constraint requires the length of $\mathbf{d}_{ij}$ to be equal to a constant d_0.

$$\Phi^{dist}(\mathbf{d}_{ij}, d_0) = (\mathbf{d}_{ij}, \mathbf{d}_{ij}) - d_0^2 = \mathbf{d}_{ij}^T \mathbf{d}_{ij} - d_0^2 = 0$$

4. *Angle* constraint requires the angle of two vectors equal to a constant α.

$$\Phi^{ang}(\mathbf{a}_i, \mathbf{a}_j) = \mathbf{a}_i^T \mathbf{a}_j - \cos\alpha = 0$$

5. *Cylindrical* constraint requires $\Phi^{cyl}(\mathbf{d}_{ij}, \mathbf{v}_i) = \mathbf{d}_{ij}^T \mathbf{d}_{ij} - (\mathbf{d}_{ij}^T \mathbf{v}_i)^2 - d_0^2 = 0$

A *spherical* constraint requires $\Phi^{S}(\mathbf{p}_i, \mathbf{p}_j) = \mathbf{p}_i - \mathbf{p}_j = 0$. Let $\mathbf{e}_1 = [1\ 0\ 0]^T$, $\mathbf{e}_2 = [0\ 1\ 0]^T$, $\mathbf{e}_3 = [0\ 0\ 1]^T$. The *spherical* constraint can be converted equivalently into 3 *dot-2* constraints $\Phi^{d2}(\mathbf{e}_k, \mathbf{p}_j - \mathbf{p}_i)(k = 1, 2, 3)$. Therefore, the *spherical* constraint can be removed from the primitives. The conversions of some usual mating conditions into primitive constraints can be found in [24].

2.2. Geometric Constraint Graph and Biconnected Property

First, the definitions of a constraint graph and its biconnected property are given as follows.

DEFINITION 1 *A constraint graph is a simple graph $G = G(V, E)$, where V and E are the sets of nodes and edges, respectively. In this graph, a node represents a rigid body, and an edge $e = (v_i, v_j)$ exists if and only if there's at least one mating condition between v_i and v_j.*

The definition of *biconnected* property of a graph is recalled from [21].

DEFINITION 2 *A connected graph $G = G(V, E)$ is biconnected if for each triple of distinct vertices u, v and w in V, there exists a path $p : u \to v$ such that w is*

not on the path p. If there is a distinct triple u, v and w such that w is on every path p : u → v, then w is called a separation node (or an articulation node) of ***G****.*

That is, once we remove an articulation node from a connected graph, the graph will be split into several biconnected components. In this paper, we note that different biconnected components of a constraint graph are independent in a sense of solving. That is, once the parts or constraints in a component are changed, the parts' positions and postures in other components will only be determined by rigid transformations if all constraints in other components are satisfied before the changes, which will save much computation.

As depicted in Fig. 1, the graph G can be decomposed into four biconnected components $G_1(v_0, v_1, v_2)$, $G_2(v_2, v_3, v_4, v_5)$, $G_3(v_2, v_6, v_7)$ and $G_4(v_7, v_8, v_9)$. Suppose v_0 be the fixed base. If the constraints represented by the edge (v_1, v_2) are changed (such as adding a new constraint, modifying the value of some design parameters), the geometric satisfaction proceeds as follows: first we recalculate the subgraph G_1 to get its configuration. Then we solve the subgraph G_2 and G_3 independently. At last G_4 is resolved. Noting that the adjacent biconnected components are connected by only one body, we can simplify the solving procedure by getting the configurations of G_2, G_3 and G_4 with rigid transformation.

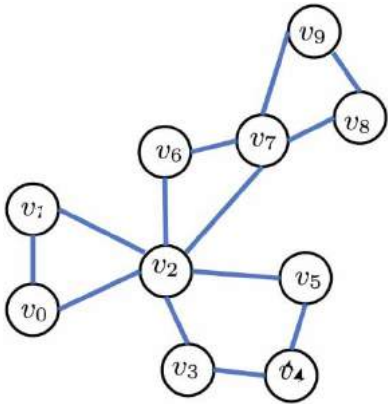

Fig. 1: A constraint graph with biconnected components.

We introduce the definition of a *propagation tree* as follows:

DEFINITION 3 *A propagation tree is a rooted tree, of which each node represents a biconnected component of a constraint graph* ***G****. A propagation tree is constructed recursively:*

1. *The root node represents the biconnected component the fixed base belongs to.*
2. *A node v_j is a child of a node v_i if the components represented by v_i and v_j share a common body, and v_i is the node of this propagation tree.*

For example, the constraint graph in Fig. 1 can be converted into a propagation tree depicted in Fig. 2.

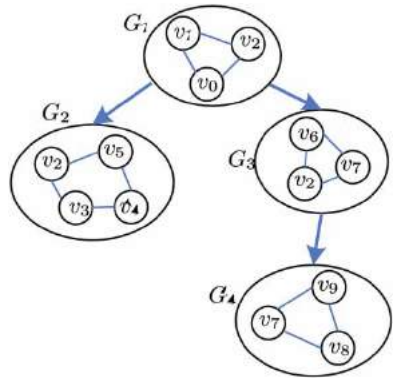

Fig. 2: The propagation tree.

3. SOLVING THE GEOMETRIC CONSTRAINT GRAPH

Based on the concepts discussed in the previous section, we propose an incremental method for solving the constraint graph. The main flow is listed as follows:

1) Construct the propagation tree from the constraint graph.
2) Identify the biconnected subgraph where the solving process will start, and solve it using the cut-joint method if closed loops exist.
3) Propagate the changes down the propagation tree using rigid transformation.

3.1. Construction of the Propagation Tree

Before building the propagation tree, we first use a biconnected decomposition algorithm presented in [21] to get all the biconnected components. An arbitrary subgraph which includes the fixed body is chosen as the root of the propagation tree. Then we construct the propagation tree recursively as follows: suppose a subgraph to be taken into the propagation tree, and then all other subgraphs adjacent to this subgraph (having an articulation node shared with it) are seemed as the children of the node corresponding to this subgraph in the propagation tree. The recursion continues until all the biconnected components are visited. The construction algorithm of a propagation tree from biconnected components is listed in Algorithm 1.

The time complexity can be analyzed as follows. Suppose the constraint graph $G = G(V, E)$. In

Algorithm 1: Constructing propagation tree

```
input : G the constraint graph
        base the base body
output: T a propagation tree
Get the set of biconnected components of G, named
SG
root(T) ← any subgraph which base belongs to ;
Set queue Q ← {root(T)} ;
while Q.NotEmpty() do
    nd ← Q.Top() ;
    for each articulation node in the subgraph nd do
        Get all other subgraphs having the
        articulation node shared with nd , named as
        children;
        Insert children into the children of nd, and
        the queue Q ;
    end
    Pop nd up from Q ;
    Remove nd from SG ;
end
return T;
```

Algorithm 1, according to [21], the biconnected decomposition algorithm can be performed in the time $O(|V|+|E|)$. The count of the biconnected components is at most $|V|$. During all the loops proceeded above, each biconnected component is visited exactly once. Thus the total time complexity of the construction algorithm is $O(|V|+|E|)$.

3.2. Solving the Biconnected Subgraph

The solving process is composed of the following steps: 1) all the geometric constraints are converted to kinematic joints; 2) multiple parallel joints are merged into a single joint; 3) a minimum spanning tree is generated and the cut constraints are solved simultaneously using a numerical method.

3.2.1. *Generalized coordinate representation of reference frames*

The pose of body i in an assembly system can be described by the origin $\mathbf{r}_i$ and an orthogonal cosine transformation matrix $\mathbf{A}_i$ from the body reference frame to the global reference frame. Using the relative generalized coordinates, the global coordinates of each body are constructed recursively. The relations of two connected bodies are depicted in Fig. 3. Let body i be the inboard of body j, (xyz) represent the global reference frame, and $(x'_k y'_k z'_k)\{k=i,j\}$ represent body reference frames, $(x''_k y''_k z''_k)\{k=i,j\}$ represent the joint reference frames fixed on each body. Orthogonal matrices $\mathbf{C}_{ij}$, $\mathbf{C}_{ji}$ and $\mathbf{A}''_{ij}$ are transformations from the joint definition frames to the body frames on bodies i and j and from the joint definition frame on body j to the joint definition frame on body i, respectively. From Fig. 3, we have:

$$\begin{aligned}\mathbf{r}_j &= \mathbf{r}_i + \mathbf{s}_{ij} + \mathbf{d}_{ij} - \mathbf{s}_{ji} = \mathbf{r}_i + \mathbf{s}_{ij} + \mathbf{A}_i\mathbf{C}_{ij}\mathbf{d}''_{ij} - \mathbf{s}_{ji}\\ &= \mathbf{r}_i + \mathbf{A}_i\mathbf{s}'_{ij} + \mathbf{A}_i\mathbf{C}_{ij}\mathbf{d}''_{ij} - \mathbf{A}_j\mathbf{s}'_{ji} \qquad (1a)\end{aligned}$$

$$\mathbf{A}_j = \mathbf{A}_i\mathbf{C}_{ij}\mathbf{A}''_{ij}\mathbf{C}^T_{ji} \qquad (1b)$$

where $\mathbf{s}'_{ij}$ and $\mathbf{s}'_{ji}$ are fixed vectors on each body frame separately. In Eqs. 1, $\mathbf{A}''_{ij}$ and $\mathbf{d}''_{ij}$ are expressed only by relative generalized coordinates $\mathbf{q}_{ij}$. For an open-loop system, there exists a unique path $0 \to 1 \to \cdots \to i-1 \to i$ from the base 0 to body i. Using Eqs. 1, $\mathbf{r}_i$ and $\mathbf{A}_i$ are derived recursively in terms of relative generalized coordinates $\mathbf{q}_{0,1}, \cdots$, and $\mathbf{q}_{i-1,i}$. In the next subsection, we will show how to construct the joint reference frame from the initial input for both the open loop system and the numerical solving phase.

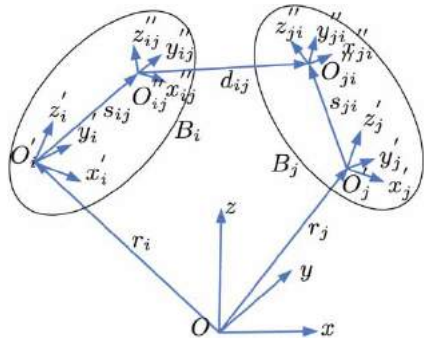

Fig. 3: The representation of relative motions of two bodies.

3.2.2. *Construction of joint reference frame from constraints*

In order to describe the relations of two adjacent bodies, body reference frame and joint reference frame need to be constructed. For an interactive CAD system, each body has its body reference frame built up by the user. When a body is brought into the assembly, as described above, its posture can be identified by a vector $\mathbf{r}_0$ pointing from the global origin to the local origin, and an orthogonal matrix $\mathbf{A}_0$.

However, when we start to solve the constraint system, to respect the user's intent as much as possible, we construct a new body reference frame and joint reference frame for the sake of solving. First, the new local reference frames of all bodies are settled down initially coincident to the global reference frame. Thus, those initial postures with respect to the global frame are zeros and identical matrices. And the joint reference frames are created from the initial positions. For example, two bodies constrained by a plane-cylinder tangency can be constructed as depicted in Fig. 4, using the procedure described in Algorithm 2, where $(\mathbf{u},\mathbf{Q})$ and $(\mathbf{v},\mathbf{P},R)$ denote the plane and the cylinder, where $\mathbf{u}$ represents the normal vector of the plane, $\mathbf{Q}$ is an arbitrary point on the plane, $\mathbf{v}$ is the vector of the axis, $\mathbf{p}$ is an arbitrary point on the axis, and R is the radius. A''_{ij} and $\mathbf{d}''_{ij}$ are defined by $A''_{ij} = \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi & \cos\phi\end{bmatrix}$, $\mathbf{d}''_{ij} = \begin{bmatrix}1 & 0\\ 0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}x\\ y\end{bmatrix}$. Let bodies 1 and 2 have original body reference frames located at (r^0_1, A^0_1) and (r^0_2, A^0_2) in the global reference frame. Body 1 has a plane P $(\mathbf{u}, \mathbf{Q})$, where

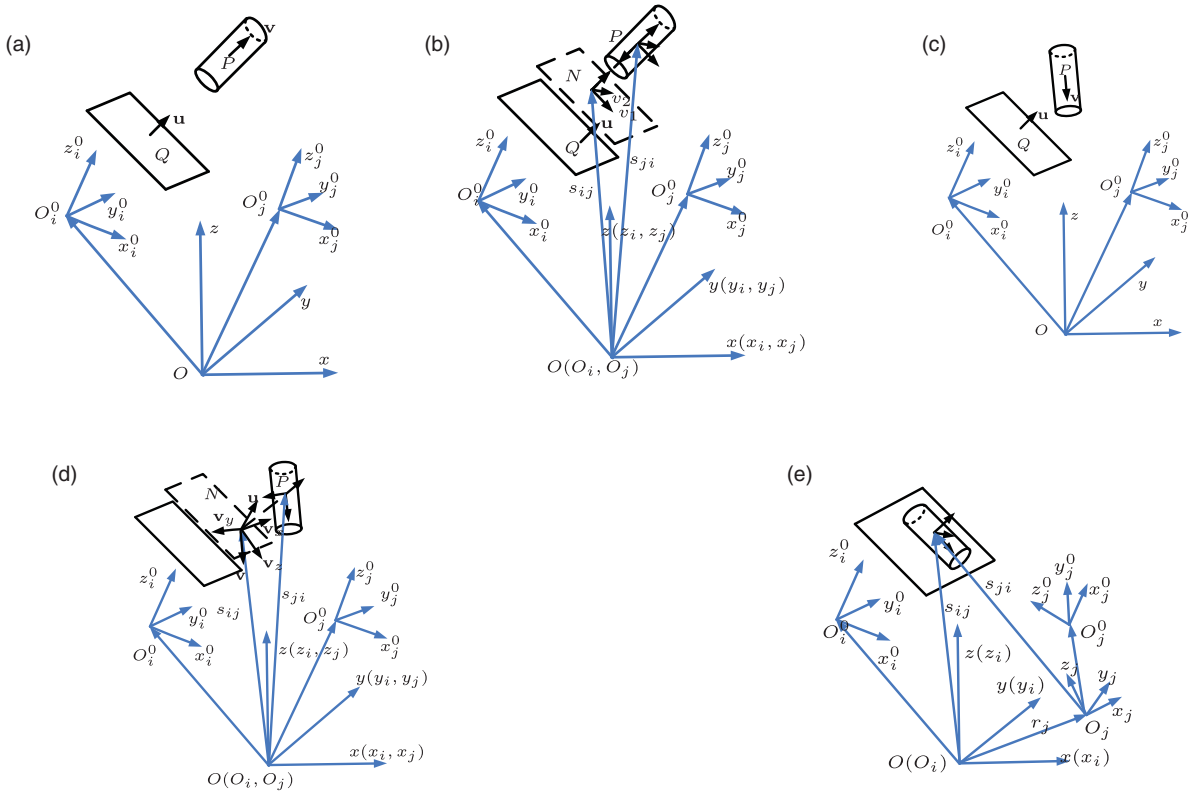

Fig. 4: The construction of joint reference frame from a plane-cylinder tangency constraint.

$\mathbf{u}' = [\,0\;0\;1\,]^T$ and $\mathbf{Q}' = [\,0\;0\;0\,]^T$. Body 2 has a cylinder $C(\mathbf{v},\mathbf{P},R)$, where $\mathbf{v}' = [\,0\;0\;1\,]^T$, $\mathbf{P}' = [\,100\;100\;100\,]^T$ and $R = 80$. Let $\mathbf{r}_1^0 = \mathbf{0}$ and $\mathbf{A}_1^0 = \mathbf{I}$, $\mathbf{r}_2^0 = [\,0\;0\;50\,]^T$ and $A_2^0 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$. Let C be in the positive side of the plane. The constructed joint reference frame is $C_{12} = C_{21} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$, $\mathbf{s}''_{12} = \begin{bmatrix} 100 \\ 100 \\ 80 \end{bmatrix}$, $\mathbf{s}''_{21} = \begin{bmatrix} 100 \\ 100 \\ 150 \end{bmatrix}$. Keep body 1 fixed, let $\mathbf{q} = 0$. From Eqs. 1, we can get the posture of constructed body reference frame of body 2: $\mathbf{r}_2 = [\,0\;0\;-70\,]^T$, $\mathbf{A}_2 = \mathbf{I}$, which means, to fulfill the tangency constraint, we just keep the posture of the body 2 invariant, and move body 2 by -70 along the global z-axis. This is an intuitional solution because the positions of the bodies change subtly.

In general, the joint reference frames are built up obeying such simple rule: letting $\mathbf{q}$ be the relative generalized coordinates, the relative postures of two connected bodies are most reasonable while $\mathbf{q} = 0$. The word *reasonable* means the changes to the assembly are as subtle as possible. Consider the cylinder-plane tangency constraint, for example. As showed above, if the cylinder and plane are parallel, their postures can be kept invariant and just translated along the normal vector of the plane to fulfill such constraint. By constructing the joint reference frame from the initial positions carefully, we can get a reasonable solution for an open-loop system analytically, instead of using the expensive and unsteady numerical optimization method in [24]. Moreover, the configuration of the constraint system at $\mathbf{q} = 0$ is a good initial guess for a closed-loop assembly problem, which facilitates the

Algorithm 2: Construction of joint frame from a plane-cylinder tangency constraint

input : $\mathbf{r}_i$, $\mathbf{A}_i$, $\mathbf{r}_2$, $\mathbf{A}_2$, $\mathbf{u}'$, $\mathbf{Q}'$, $\mathbf{v}'$, $\mathbf{P}'$, R, sign
output: $\mathbf{s}''_{ij}$, $\mathbf{C}_{ij}$, $\mathbf{s}''_{ji}$, $\mathbf{C}_{ji}$
$\mathbf{u} \leftarrow \mathbf{A}_i\mathbf{u}'$
$\mathbf{v} \leftarrow \mathbf{A}_2\mathbf{v}'$
$\mathbf{Q} \leftarrow \mathbf{A}_i\mathbf{Q}' + \mathbf{r}_i$
$\mathbf{P} \leftarrow \mathbf{A}_2\mathbf{P}' + \mathbf{r}_2$
$\mathbf{M} \leftarrow \mathbf{Q} + sign * R * \mathbf{u}$
$\mathbf{s}''_{ij} \leftarrow \mathbf{P} - (\mathbf{P} - \mathbf{M}, \mathbf{u})\mathbf{u}$
$\mathbf{s}''_{ji} \leftarrow \mathbf{P}$
if $\boldsymbol{u} = -\boldsymbol{v}$ **then**
  $\mathbf{v} \leftarrow -\mathbf{v}$
end
if $\boldsymbol{u} = \boldsymbol{v}$ **then**
  get two unit vectors $\mathbf{v}_i$, $\mathbf{v}_2$ which make $\mathbf{v}_i$, $\mathbf{v}_2$ and $\mathbf{u}$ construct a right-hand coordinates.
  $\mathbf{C}_{ij} \leftarrow [\mathbf{v}_i, \mathbf{v}_2, \mathbf{u}]$
  $\mathbf{C}_{ji} \leftarrow [-\mathbf{u}, \mathbf{v}_i, \mathbf{v}_2]$
else
  $\mathbf{v}_z \leftarrow \mathbf{v} - (\mathbf{u}, \mathbf{v})\mathbf{u}$
  $\mathbf{v}_z \leftarrow \mathbf{v}_z / \|\mathbf{v}_z\|$
  $\mathbf{v}_y \leftarrow \mathbf{v}_z \times \mathbf{u}$
  $\mathbf{v}_x \leftarrow \mathbf{v}_y \times \mathbf{v}$
  $\mathbf{C}_{ij} \leftarrow [\mathbf{u}, \mathbf{v}_y, \mathbf{v}_z]$
  $\mathbf{C}_{ji} \leftarrow [\mathbf{v}_x, \mathbf{v}_y, \mathbf{v}]$
end
return $\mathbf{s}''_{ij}$, $\mathbf{C}_{ij}$, $\mathbf{s}''_{ji}$, $\mathbf{C}_{ji}$

possible numerical method to solve the close-loop system. The solution closest to the initial input is given constantly.

3.2.3. Merging the joints of two connected bodies

Since multiple constraints would possibly be added into two connected bodies, after each constraint converted into a joint, there will be multiple edges in two connected nodes. Unlike the direct numerical solving method proposed in [24], we try to merge the parallel edges into single edge analytically, which will reduce the number of generalized coordinates and non-linear equations to be numerically solved. For example, suppose that two coplanar constraints will be added between two bodies. If we convert each constraint into a plane joint, there will be two planar pairs, each of which has three generalized coordinates. With the cut-joint method discussed later, a planar pair will be cut, and three non-linear equations must be solved. However, we know that two coplanar constraints can be merged into a prismatic pair, which only has one generalized coordinates. This will dramatically reduce the number of non-linear equations to be solved simultaneously, for a closed-loop constraint system. Our geometric reasoning exhaustedly enumerates the pairs of joints to merge them into one single joint, until only one joint exists. If no more pair of joints can be merged, we select the joint having a minimum number of generalized coordinates as the joint of the two bodies, and convert all residual joints into equivalent primitive constraints, which will be solved numerically combined with other constraints in the next subsection.

3.2.4. Generation of a maximum weight spanning tree

For a closed-loop system, we adopt the cut-joint method used in [1,6,17,24,25]. The cut-joint method removes some edges from the constraint graph to form a spanning tree. The cut constraints and those unable to be reasoned should be solved simultaneously. Let e_i be any edge, $reasoned(e_i)$ represent the number of reasoned constraints, and $residual(e_i)$ represent the number of constraints unable to be reasoned. Let n be the count of the equations required to be solved. C and E represent the sets of cut and all edges, respectively. Thus, we have the following equation:

$$n = \sum_{e_i \in C} reasoned(e_i) + \sum_{e_i \in E} residual(e_i),$$

in which the second term is invariant. To minimize n, we just need to minimize the first term. If we take $reasoned(e_i)$ as the weight of each edge, generating a maximum weight spanning tree will minimize n. Since the count of the system's residual DOFs are constant, the minimization of n means the minimization of the number of the unknowns of the equations, too.

After generating the maximum weight spanning tree, the residual equations to be solved can be written as $\Phi(\mathbf{q}) = \mathbf{0}$, where $\mathbf{q}$ is the collection of generalized coordinates of the spanning tree. To solve this set of equations, a numeric iteration method is indispensable. As stated in the previous subsection, the iteration starts constantly from the initial value $\mathbf{q}_0 = 0$ in which our construction of joints implies that the configuration of the assembly at $\mathbf{q}_0 = 0$ is more appropriate intuitionally. The numerical solving method of these equations can be found in [15,24].

Algorithm 3: Propagating algorithm

```
input  : T: the propagation tree
         sg: the affected biconnected component
output: The solved result
Solve the biconnected component as depicted above;
Set queue Q ← {sg} ;
while Q.NotEmpty() do
    nd ← Q.Top() ;
    for each child c of nd do
        if the base of c is changed then
            Let B_0 and B_0', r_0 and r_0' be the global
            rotation matrices and origins, of the base
            before and after the changes;
            for each body B_i in c do
                B_i' ← B_i B_0'^T B_i;
                r_i' ← r_i + r_0' − r_0;
            end
            Insert c into the queue Q;
        end
    end
    Pop c up from the queue Q;
end
```

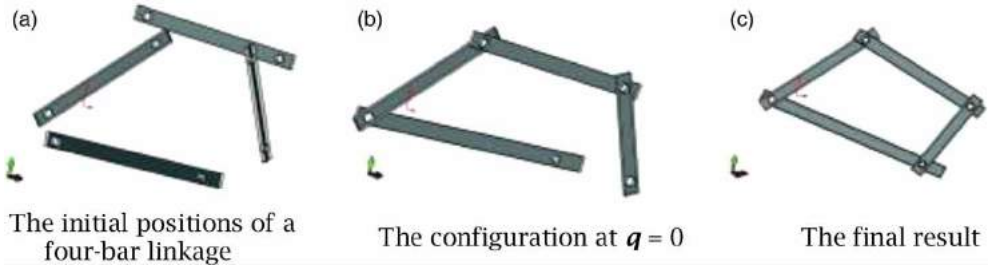

Fig. 5: The solving process of a four-bar linkage.

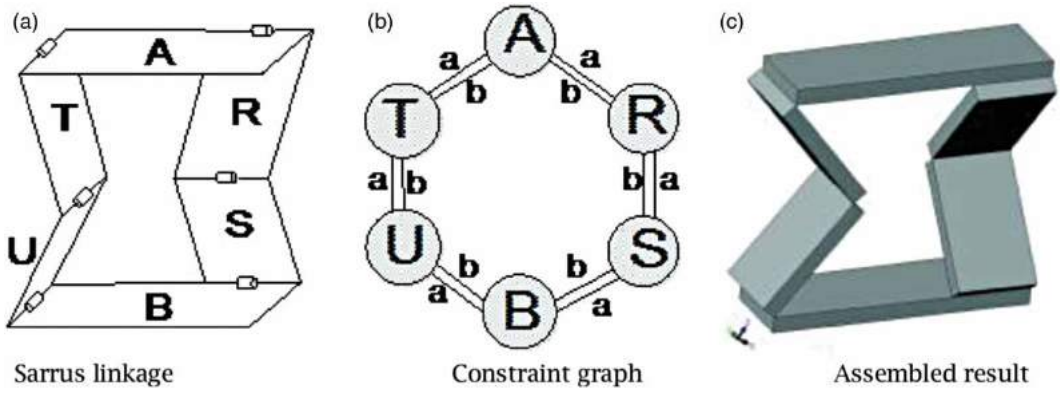

Fig. 6: A Sarrus linkage and its constraint graph.

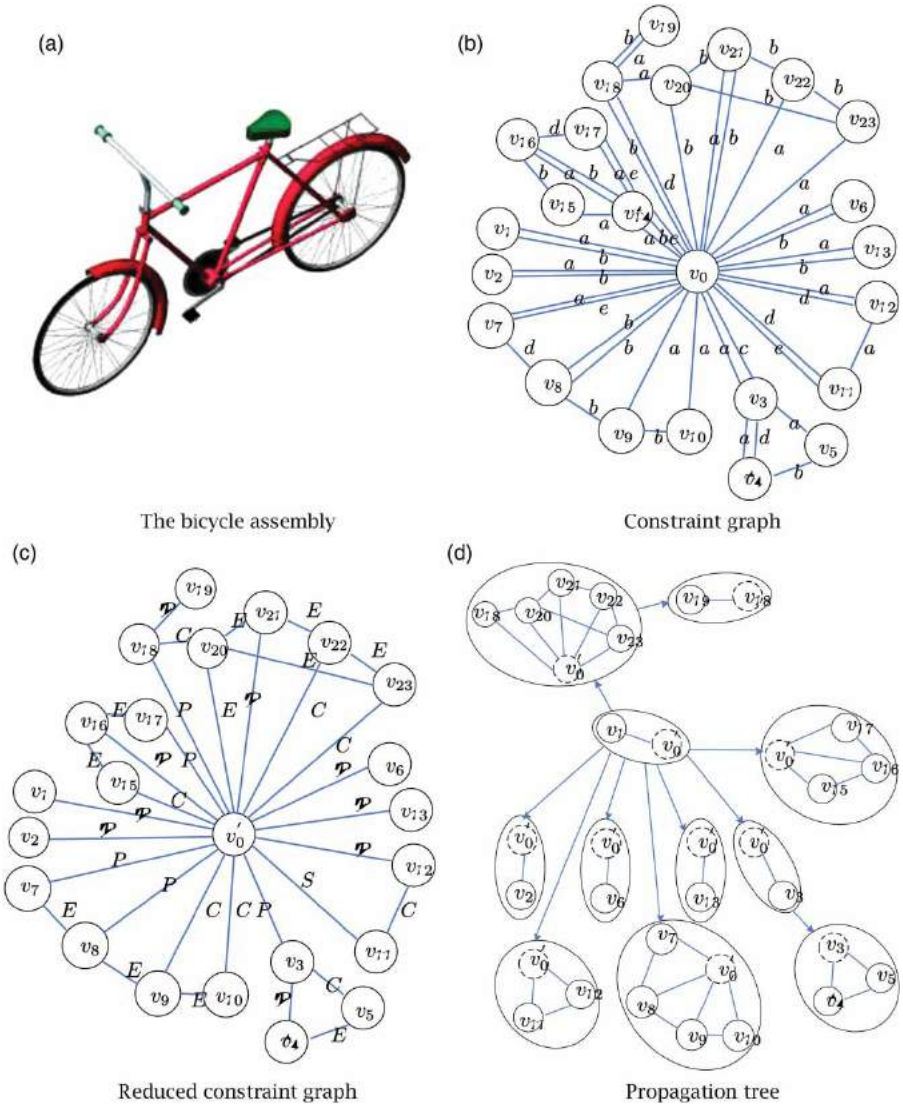

Fig. 7: A bicycle assembly and its constraint graph.

3.3. Propagation of geometric constraints solving

Once the affected biconnected component is recalculated, the changes are propagated down on the propagation tree. The propagating algorithm is listed in Algorithm 3. The propagating process is quite straightforward. We use the breadth-first method to walk through the entire tree. If the global reference frame of the base of a subgraph keeps unchanged, the reference frames of all the bodies in the subgraph and its children don't need any adjustment. The procedure can be proved correct easily. Substituting the related B_i' and $\mathbf{r}_i'$ into the primitive constraints, we can verify that all the equations still hold.

Joint	Abbr.	Specification	DOC
Cylindrical	C	two axes are aligned	4
Planar	E	two surfaces are coincident	3
Revolute	R	one rotary motion	5
Prismatic	P	one translational motion	5
Slide	S	two translational motion	4

Tab. 1: Assembly constraints used in the example.

4. EXAMPLES

The proposed method has been implemented as a 3D assembly constraint solver in interactive CAD software. Fig. 5 illustrates the solving process of a four-bar linkage. Fig. 5(a) is the initial guess input by the user. Adopting the strategy depicted in Section 3, Fig. 5(b) gives the configuration of the spanning tree at $\mathbf{q} = 0$ with a revolute joint cut. At last, the final configuration of the linkage is given in Fig. 5(c). It takes only several iterations to get a result near to the initial input. A realization of Sarrus linkage is depicted in Fig. 6. The Sarrus linkage composed of only rotational joints has a rectilinear motion, vertically up and down. Fig. 6(a) is the demonstration of a Sarrus linkage. After solving the geometric constraint graph depicted in Fig. 6(b), the final configuration is obtained as Fig. 6(c). Fig. 7 gives an example of a bicycle assembly which involves 24 parts, and contains 50 geometric constraints which can be converted into 168 primitive constraints, designated by the designer. Table 1 lists the constraints used in the example. Table 2 gives all the joints used in Fig. 7(a). The original constraint graph is given in Fig. 7(b). Fig. 7(c) illustrates the reduced constraint graph after the construction and merging of the joints between two connected parts. For an entire computing from scratch, Fig. 7(b) requires solving 75 equations, of which the maximum number of equations to be solved simultaneously is 18, but Fig. 7(c) only requires solving 35 equations, of which the count of equations to be solved simultaneously is 12. Fig. 7(d) depicts the propagating process of the constraint graph. If the user modifies the assembly interactively, say, tries to add a plane-plane distant constraint between the parts v_0' and v_3, the solver identifies the subgraph (v_0', v_3), and solves it. The changes can be propagated out only by rigid transformation. As the joints are created obeying the strategy mentioned previously, the solution often gives a configuration of the assembly varying from the original configuration as subtly as possible.

Constraint	Abbr.	Specification	DOC
CoLine	a	two lines are superposed	4
CoPlane	b	two planes are coplanar	3
ParPlane	c	two planes are parallel	2
DistPP	d	distance constraint between planes	3
PerpLL	e	two lines are perpendicular	1

Tab. 2: Spatial joints used in the example.

5. CONCLUSIONS AND DISCUSSIONS

In this paper, an incremental method is proposed to solve the assembly constraints especially in the under-constraint state in an interactive CAD system. During the assembling process, a designer frequently adds or removes some constraints, changes the mating parameter, and maneuvers some body. Meanwhile, such constraint graph normally mingles open-loop with closed-loop subgraphs. The proposed method takes the following steps. First, identifies the biconnected component where the changes occurred and solves it using cut-joint method. Then propagate such changes out by rigid transformation. With such a strategy adopted, the geometric constraints of other biconnected components don't need to be solved. Before using the cut-joint method to solve the biconnected component, we first use the geometric reasoning to get an optimum relative generalized coordinates to reduce the number of constraints required to cut. Also, we propose a strategy to construct the joint reference frames, with the initial positions of the bodies considered, which facilitates both solving open-loop and closed-loop subsystems. By adopting a maximum weight spanning tree, we prove that the count of the cut constraints is minimized. Although a cut-joint method is adopted to solve the biconnected component in this paper, other methods to solve geometric constraint problems can be considered to solve this subproblem if the number of cut constraints is too large to solve efficiently in a numerical iterative method. A graph-constructive method

may help to decompose the biconnected component into some fewer subproblems further. The comparison of efficiency and steadiness of cut-joint methods with those constructive methods is our future work.

ACKNOWLEDGEMENTS

The research was supported by Chinese 973 Program (2010CB328001) and Chinese 863 Program (2012AA040902). The first author was supported by the NSFC (61035002, 61272235). The second author was supported by the NSFC (61063029, 61173077). The third author is also a senior visiting professor at Jiangxi Academy of Sciences.

REFERENCES

[1] Bae, D.-S.; Haug, E.-J.: A Recursive Formulation for Constrained Mechanical System Dynamics: Part II - Closed Loop Systems, Mechanics of Structures and Machines, 15(4), 1987, 481–506.

[2] Borning, A.: The Programming Language Aspect of ThingLab, ACM Transactions on Programming Language and Systems, 3(4), 1981, 353–387.

[3] Fudos, I.; Hoffmann, C.-M.: A Graph-constructive Approach to Solving Systems of Geometric Constraints, ACM Transactions on Graphics, 16(2), 1997, 179–216.

[4] Gao, X.-S.; Jiang, K.; Zhu C.-C.: Geometric Constraint Solving with Conics and Linkages, Computer-Aided Design, 34(6), 2002, 421–433.

[5] Gao, X.-S.; Lin, Q.; Zhang, G.-F.: A C-tree Decomposition Algorithm for 2D and 3D Geometric Constraint Solving, Computer-Aided Design, 38(1), 2006, 1–13.

[6] Haug, E.-J.: Computer Aided Kinematics and Dynamics of Mechanical Systems: Basic Method, Allyn and Bacon, Boston, 1989.

[7] Kim, J.-S.; Kim, K.-S.; Lee, J.-Y.; Jung, H.-B.: Solving 3D Geometric Constraints for Closed-loop Assemblies, International Journal of Advanced Manufacturing Technology, 23(9-10), 2004, 755–761.

[8] Kim, S.-H.; Lee, K.: An Assembly Modelling System for Dynamic and Kinematic Analysis, Computer-Aided Design, 21(1), 1989, 2–12.

[9] Kramer, G.-A.: A Geometric Constraint Engine, Artificial Intelligence, 58(1-3), 1992, 327–360.

[10] Laman, G.: On Graphs and Rigidity of Plane Skeletal Structures, Journal of Engineering Mathematics, 4(4), 1970, 331–340.

[11] Latham, R.-S.; Middleditch, A.-E.: Connectivity Analysis: A Tool for Processing Geometric Constraints, Computer-Aided Design, 28(11), 1996, 917–928.

[12] Lee-St J. A.; Sidman J.: Combinatorics and the rigidity of CAD systems. Computer-Aided Design, 45(2), 2013, 473-482.

[13] Li, Y.-T.; Hu, S.-M.; Sun, J.-G.: A Constructive Approach to Solving 3-D Geometric Constraint Systems Using Dependence Analysis, Computer-Aided Design, 34(2), 2002, 97–108.

[14] Light, R.; Gossard, D.: Modification of Geometric Models through Variational Geometry, Computer-Aided Design, 14(4), 1982, 209–214.

[15] Liu, Y.; Song, H.-C.; Yong, J.-H.: Calculating Jacobian Coefficients of Primitive Constraints with Respect to Euler Parameters, International Journal of Advanced Manufacturing Technology, Accepted. doi: 10.1007/s00170-012-4643-9.

[16] Mathis, P.; Thierry, S.-E.-B.: A Formalization of Geometric Constraint Systems and Their Decomposition, Formal Aspects of Computing, 22(2), 2010, 129–151.

[17] Oliver, J.-H.; Harangozo, M.-J.: Inference of Link Positions for Planar Closed-loop Mechanisms, Computer-Aided Design, 24(1), 1992, 18–26.

[18] Owen, J.-C.: Algebraic Solution for Geometry from Dimensional Constraints, in: Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications, 1991, 397–407.

[19] Peng, X.; Lee, K.; Chen, L.: A Geometric Constraint Solver for 3-D Assembly Modelling, International Journal of Advanced Manufacturing Technology, 28(5-6), 2006, 561–570.

[20] Schreck, P.; Schramm, É.: Using Invariance under the Similarity Group to Solve Geometric Constraint Systems, Computer-Aided Design, 38(5), 2006, 475–484.

[21] Tarjan, R.: Depth-first Search and Linear Graph Algorithms, SIAM Journal on Computing, 1(2), 1972, 146–159.

[22] Tay T.-S.: Rigidity of multi-graphs. I. linking rigid bodies in n-space. Combinatorial Theory Series B 26, 1984, 95–112.

[23] van der Meiden, H.-A.; Bronsvoort, W.-F.: A Non-rigid Cluster Rewriting Approach to Solve Systems of 3D Geometric Constraints, Computer-Aided Design, 42 (1), 2010, 36–49.

[24] Xia, H.-J.; Wang, B.-X.; Chen, L.-P.; Huang, Z.-D.: 3D Geometric Constraint Solving Using the Method of Kinematic Analysis, International Journal of Advanced Manufacturing Technology, 35 (7-8), 2008, 711–722.

[25] Zou, H.; Abdel-Malek, K.; Wang, J.-Y.: Computer-Aided Design Using the Method of Cut-joint Kinematic Constraints, Computer-Aided Design, 28(10), 1996, 795–806.