Detecting Minimum Over-constrained Subgraphs in 2D and 3D Based on Degree of Freedom Analysis

H. L. Zou[1] and Y. T. Lee[2]

[1]Nanyang Technological University, zouhongliu@pmail.ntu.edu.sg
[2]Nanyang Technological University, mytlee@ntu.edu.sg

ABSTRACT

We present a flow-based method for finding minimum over constrained subgraphs in a geometric constraint graph. In 2D several different approaches have been implemented, while the 3D problem, in which the entities are points, lines and planes of a model, is much less investigated in the literature. We present a general algorithm for the problem both in 2D and 3D, show that the algorithm is correct and can be used to identify if there is an over constrained subgraph in a constraint graph.

Keywords: geometric constraints, over constrained subgraph, degree of freedom, degree of rigidity

1. INTRODUCTION

Geometric constraint solving, especially in 3D, has many applications, such as in CAD, kinematics, reverse engineering, molecular modeling and robotics [1]. One active research area in CAD concerns computer-aided conceptual design, whereby a designer sketches a design in 2D, and a 3D model is recovered automatically from the sketch [2-4]. Such 2D sketches carry no dimensions, and the dimensions in the recovered 3D model are based on the pixel positions of the lines and vertices in the sketch, which are inaccurate. Usually such recovered dimensions bear no resemblance to the actual sizes the designer had in mind. Hence one key problem after the recovery of a 3D object is to give it proper dimensions. A similar problem arises in reverse engineering when a 3D model is recovered from a point cloud [5]. It is neither practical nor desirable to require the user to provide the dimension for every entity – vertex, edge or face – present. One solution is to set up the constraints between the entities, and resolve the dimensions based on these constraints computationally. Some of the constraints can be established automatically, but there will always be some that must be given by the user. Clearly, it is desirable to minimize the user input on such a potentially tedious and error-prone task.

Whether assigned manually or automatically, it is necessary to determine if a set of constraints is sufficient for describing a model completely. There can be over constraint or under constraint, and there may also be redundancies, which include structural and numerical redundancies. A structural redundancy over constrains the system. For instance, the constraint $f(x_1, x_2) = 0$ which constrains two variables x_1 and x_2 in a system will lead to a structural redundancy if two other constraints $g_1(x_1, x_2) = 0$ and $g_2(x_1, x_2) = 0$ are present, since the values of x_1 and x_2 are implicitly determined already by g_1 and g_2. Constraints and the entities they constrain can be represented as a graph, and f, g_1, and g_2 result in an over-constrained subgraph. The problem can be rectified by discarding one of the constraints. A system can be numerically redundant or inconsistent. For example, two constraints expressed by $x_1+x_2=1$ and $x_1+x_2=0$ are inconsistent; $x_1+x_2=1$ and $2x_1+2x_2=2$ are redundant.

A constraint system can be expressed as a system of equations [6]. If structural redundancy exists in the system, then the Jacobian matrix of the system of equations is singular. Light [6] identified an invalid dimensioning scheme by singularity of the Jacobian matrix, but a singular Jacobian matrix could also be produced by structural or inconsistent redundancies. The complexity of the method to resolve the redundancy is $O(N^3)$ or worse [7]. Buchanan [8] determined whether a system of equations is inconsistent by using the Grobner basis. Gao [9] gave a complete method for deciding whether a set of constraints is independent and whether the constraint system is numerically inconsistent based on Wu-Ritt's decomposition algorithm. Despite the advantages of the algebraic approaches using Grobner bases or the Wu-Ritt method, both of them require exponential time complexity. It is not uncommon for them to take some tens of minutes or even hours, which is not acceptable in a real time interactive system.

Numerous works have addressed the problem of structural redundancies. In 2D, the triangle decomposition method was used by many researchers in geometric constraint solving [9-11], so their methods for identifying over-constrained subgraphs were based on triangle decomposition too. For example, in Fudos's method [13-15], if two well-constrained clusters share more than one geometric element, then over-constraint is detected. But the method can be used only within a limited domain, and cannot be applied in 3D.

Latham [16] proposed a method based on the maximum b-matching algorithm to decompose a constraint problem into a sequence of solvable subgraphs. The crux of this algorithm is the detection and correction of over-constraints and under-constraints. But this method of detection does not work well in some cases. For example in the case depicted in Fig. 2(a), this method cannot detect the over-constrained subgraph $p_2p_3p_4p_5$. Maximum b-matching is a special case in generalized maximum matching (MM), of which Hoffmann stated [17], "the MM method may or may not detect over-constrained subgraphs, depending on the initial choice of vertices for reducing weights". He proposed a method MM1 to correct this drawback, but did not deal with 3D cases.

Li's method can detect over-constraint in 3D cases [18], but all the entities in the constraint graph must have six degrees of freedom. So his algorithm cannot be used in cases where the entities are points, lines and planes, which do not have six degrees of freedom. Recently, Langbein [5] proposed a method of identifying over-constraint based on Kramer's degree of freedom analysis [19] and Li's method of analyzing dependencies between geometric objects. But his method did not have a rigorous proof, and he could not obtain the minimum over-constrained subgraph. Jermann proposed a new concept of extensive structure rigidity [20, 21], which is useful and will be used in our algorithm. His algorithm may find all the over-rigid subgraphs, but the number of sub-objects (called DOR-minimals in the paper) the algorithm has to deal with is large, even though it may stop early without utlising all the sub-objects. Jermann's paper shows that Hoffmann's algorithm for identifying dense subgraphs may fail because of geometric rigidity [22].

Deciding whether a constraint system contains structurally redundant constraints and identifying over-constrained subgraphs are key problems in geometric constraint-solving. This paper presents a new algorithm to find the minimal over-constrained subgraph in a constraint system and identify the existence of structural redundancy, thus allowing the user to take action to keep the system properly constrained. The rest of this paper concentrates on structural properties of a constraint system and ignores numerical redundancy.

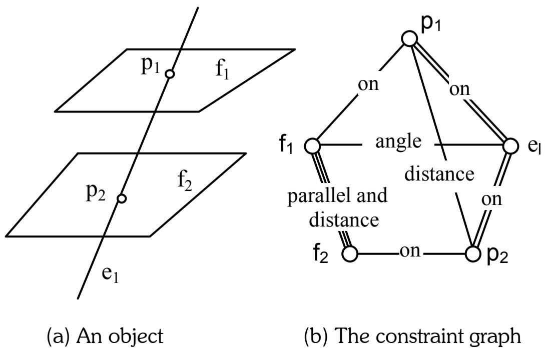
2. FINDING OVER-CONSTRAINED SUBGRAPHS

Finding over-constrained subgraphs in a constraint system is concerned with the association between geometric elements and their constraints. In this paper, the analysis is presented mainly in the 3D domain; 2D cases are simpler. This section presents the algorithm for identifying over-constrained subgraphs. But before that, there are some essential fundamental definitions.

2.1 Definitions and graphical representation

A **geometric constraint system** can be represented by a constraint graph G=(V, E), where V is the set of nodes representing the entities, including points, lines and planes. E is the set of arcs representing constraints between the entities. For example, Fig. 1(b) is the constraint graph of the constraint system shown in Fig. 1(a). The V of its constraint graph is composed of two planes f_1, f_2, two points p_1, p_2, and one line e_1. The E of the constraint graph is composed of parallel and distance constraints between f_1 and f_2, the angle between e_1 and f_1, the distance between p_1 and p_2, with p_1 lying on f_1, p_2 lying on f_2, p_1 lying on e_1 and p_2 lying on e_1.

Finding the over-constrained subgraph is a case of analyzing the degrees of freedom in the system.

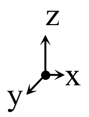
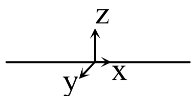
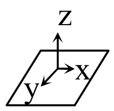


(a) An object (b) The constraint graph

Fig. 1. A 3D example

The **weight of an entity** is equal to the degrees of freedom (DOF) of the entity.

As shown in Table 1, in 3D, the weight of a point is three, for the three translations in each of the three spatial dimensions x, y and z; i.e. the point is free to move in 3D. The weight of a line is four, for two translations along two axes orthonormal to the line, and two rotations about these two orthonormal axes. The weight of a plane is three, for one translation along the plane normal and two rotations about the axes orthonormal to the plane normal.

Entity	Local coordinate system	Freedom of translation	Freedom of rotation
point		x axis y axis z axis	none
line		y axis z axis	y axis z axis
plane		z axis	x axis y axis

Tab. 1. Freedoms of geometric entities in 3D

The **weight of a constraint** is the number of DOF eliminated by the constraint, which is the number of equations required to define that constraint. For example, the constraint of distance between two points requires one equation that eliminates one degree of freedom, thus its weight is 1.

The relationships between different pairs of basic geometric entities, the constraints between them and the degrees of freedom these constraints consume in 3D are listed in Table 2.

Entity 1	Entity 2	Constraint type	Constraint weight	DOR of rigid graph
point	point	distance between two points	1	5
point	line	distance between point and line	1	6
		point lies on line	2	5
point	plane	distance between point and plane	1	5
		point lies on plane	1	5
line	line	angle between two intersecting lines	1	6
		parallel & distance between two lines	3	5
line	plane	parallel & distance between line and plane	2	5
		angle between line and plane	1	6
		line lies on plane	2	5
		perpendicular between line and plane	2	5
plane	plane	angle between two planes	1	5
		parallel & distance between two planes	3	3

Tab. 2. Basic constraint types

The **DOF of a graph** G is equal to the difference between the sum of the weights of the entities in the graph and the sum of the weights of its constraints.

A **rigid subgraph** is a well-constrained subgraph, in which the relative positions of the entities are fixed. In Table 2, the entity-entity pair plus the constraint in each row forms a rigid graph. The **degree of rigidity (DOR) of a subgraph** is the number of independent displacements it admits, i.e. the DOF of the corresponding rigid subgraph. Hence the DOR of a rigid subgraph is the same as its DOF. DOR depends on the geometric properties of the entities and the constraints [21]. For example, the DOR of a line (a pair of points and the distance between them, which form a rigid subgraph) is 5 in 3D, and the DOR of two parallel planes in 3D is 3.

In the case of 2D planar constraint problems, the entities are points and lines. The weights of all the entities are two, and all the constraints one.

In Fig. 2(a), the constraint graph is composed of five 2D points and seven point-point distances identified by the lines between them. Subgraph $p_2p_3p_4$ is rigid (see Fig. 2 (b)), and its DOF is 3, so the DOR of subgraph $p_2p_3p_4$ is 3. In Fig. 2(c), though the DOF of $p_2p_3p_4p_5$ is 2, the DOR of subgraph $p_2p_3p_4p_5$ is 3, because the DOF of its corresponding rigid subgraph $p_2'p_3'p_4'p_5'$ is 3. In Fig. 2(d), the DOR of subgraph $p_1p_2p_5$ is 3 though the DOF of $p_1'p_2'p_5'$ is 4.

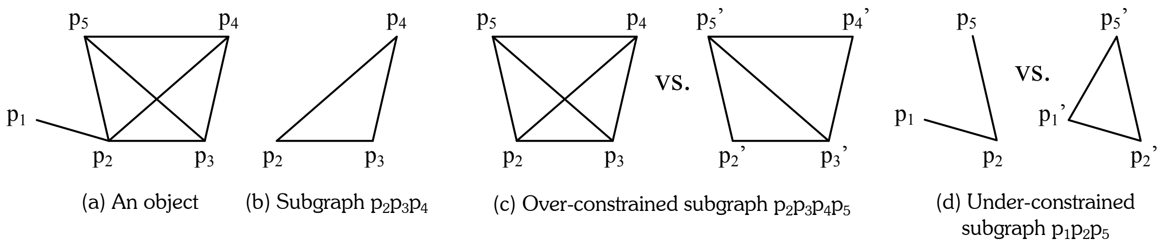


Fig. 2. A 2D example on degree of rigidity

If the DOF of a subgraph is less than its DOR, the subgraph is **over-constrained**.

In Table 2, all the subgraphs consisting of two entities and one constraint are rigid, since the relative positions between the two entities cannot be changed, i.e. the subgraphs are not deformable.

2.2 Property of rigid subgraphs

Property of DOR computation [21]: if O_1 and O_2 are two rigid subgraphs, and $O_1 \subset O_2$, then DOR $(O_1) \leqslant$ DOR (O_2). Because DOR represents the relative displacements allowed in a subgraph, and adding objects to it cannot remove the displacements, it means DOR may increase but cannot decrease with increasing number of objects.

The section below analyses the DOF of different rigid subgraphs. The results will be applied to the algorithm of identifying over-constrained subgraphs.

Theorem 1 In 3D cases, the DOR of any nontrivial subgraph is 3, 5 or 6.

Proof

Table 2 lists the types of constraint between two basic entities in 3D, and all of them are rigid. The DOF of the graphs, which consist of two entities and one constraint in the table are 3, 5 or 6. By adding entities and constraints to the basic graphs in Table 2, we can obtain new rigid graphs. If the basic graph contains two parallel planes and the distance between them, then the DOF of the rigid graph is 3. After adding another parallel plane and its distance from one of the existing planes, the DOF of the new rigid graph remains at 3, because in the new rigid graph the sum of the weights of the three nodes is 9 and the sum of the weights of the two constraints is 6 (parallel and distance constraints between two planes). So this gives DOF = 3 to the new rigid graph. This is the only case where the DOF of a new rigid graph is 3. If we add other types of entity and constraint to a graph with DOF=3 to form a new rigid graph, the new graph then contains other types of subgraphs, whose DOR may be 5 or 6 (see Table 2). According to the property of DOR computation, the new graph's DOR is 5 or 6.

When the DOF of the basic graph in Table 2 is 5, after adding entities and constraints to the basic graph to obtain a new rigid graph, according to the property of DOR computation, we can only obtain a new rigid graph with DOR$\geq$ 5, which means either 5 or 6.

After adding entities and constraints to a basic graph with DOF=6, the new rigid graph's DOR may be $\geq$ 6, according to property of DOR computation. However, the maximum number of displacements possible in a valid 3D model (i.e. a rigid graph) is 6. So the new rigid graph's DOR has to be 6.

From the analysis above, it is clear that the DOR of any rigid subgraph is 3, 5, or 6. ■

It is easy to identify a rigid subgraph with DOR=3 in a constraint graph. The subgraph is a set of parallel planes with known distances between them. We will focus on identifying rigid subgraphs with DOR=5 in a constraint graph by analyzing the properties of its entities and constraints. There are four different types of rigid subgraph with DOF=5; they are discussed below.

1. For a rigid subgraph with DOF=5 containing a point and a line, the point must lie on the line.

As stated before, a point has DOF = 3 and a line has DOF = 4. A non-rigid graph containing a line and a point unconstrained will have DOF=7. If the distance between the line and the point is given, the system becomes a rigid graph with DOF=6. For the DOF to be 5, we need to remove the rotational freedom of the point about the line by requiring that the point lies on the line.

2. For a rigid subgraph containing one point, one plane and the perpendicular distance from the point to the plane, the DOR=5. In the case of multiple points and multiple planes, for DOR=5, all the planes must be parallel and the points must lie in one line perpendicular to the plane.

A plane has DOF=3 and a point also has DOF=3. An unconstrained system with a plane and a point has DOF=6. Specifying the distance between the plane and the point reduces the DOF by one, hence such a system has DOF=5. Adding another point to the system increases its DOF to 5+3=8. Fixing the distance between this new point and the plane reduces the DOF to 7. Further requiring that this point lies on the plane normal containing the first point reduces the DOF by another 2 (the distance between the two points and the rotation of the second point about the said normal). Hence, for a system with a plane and two points, the DOF=5 when the points lie in the same normal to the plane. The same is true when there are more than two points. Further it can be shown that if there are multiple planes, then they must be parallel to maintain DOF=5, since the DOF of a system of parallel planes with known distances between them is 3.

3. For a rigid subgraph with DOF=5 containing a line and a plane, the line must be perpendicular to the plane or parallel and at a given distance to the plane.

A line has DOF=4, and a plane has DOF=3; thus an unconstrained system with these two entities has DOF=7. Constraining the line to be perpendicular to the plane removes two freedoms: the rotation of the line about the plane normal through their point of intersection, and the translation of the line along this normal. Hence the DOF is reduced to 5. In the parallel case, the line being parallel to the plane and the distance between them reduce the DOF by 2, and hence the DOF of the resulting constraint system is also 5.

4. Except for the cases listed in Table 2, there are no other rigid subgraphs with DOF=5 which have only points or lines or planes.

Based on the discussion above, we can find a rigid subgraph with DOF=5 by the types of entity and constraint in a given graph. The case of DOR=6 will not be discussed here as it is will be seen in the algorithms below that there is no need to deal with such cases directly.

2.3 Algorithm for finding a minimum over-constrained subgraph

The algorithm starts with a subgraph G' of G; initially G' includes only all the entities and no constraint. Therefore, we know G' is under-constrained, and there is no over-constrained subgraph in G'. We then add the constraints of G to G' one by one. When a constraint *e* is added to G', we attempt to find the over-constrained subgraph, starting from the two end points, *a* and *b*, of *e*.

When the weight of e=1, if there is no rigid subgraph with DOF=5 or 6 in G' (which does not contain *e*) including the two end points, then no over-constrained subgraph is found. Otherwise a subgraph *G''* would be identified by Algorithm 2, and {*G'', e*} is the over-constrained subgraph. The DOF of *G''* may be 5 or 6 when the weight of *e* is 1, according to Theorem 1. Fig. 3 shows a sketch of the G', *G''*, base graph and *a*, *b*. The base graph begins with {a, b}; some of the elements of G may be added into it in Algorithm 1, then {base graph, *e*} represents an acceptable rigid subgraph whose DOF maybe 3 or 5. When the weight of e=2, Algorithm 2 will identify a subgraph *G''* with DOF 5, 6 or 7, if it exists. And when the weight of e=3, we need to find a subgraph G'' with DOF less than 8.

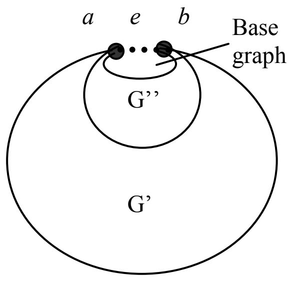


Fig. 3. Variables in the algorithm

It is easy to detect a subgraph with weight 3, as only a series of parallel planes can have DOF=3. We can detect such a situation by searching the groups of parallel planes, two of which are end points of *e*.

Algorithm 1: Finding a minimum over-constrained subgraph

```
1.   G'={V}
2.   for every constraint e in G do
3.       base graph=e and a, b which are two end nodes connected by e in G'
4.       G''={∅}
5.       if can identify a rigid subgraph G'' with DOF=5 including base graph
6.           go to line 19
7.       if can identify a subgraph G'' with DOF=6 including base graph
8.           if {G'',e} is one case of rigid subgraphs with DOF =5, and weight of e is 1
9.               add G'' to base graph, go to line 7
10.          else go to line 19
11.      if can identify a subgraph G'' with DOF=7 including base graph and weight of e is 2
12.          if {G'',e} is one case of rigid subgraphs with DOF =5
13.              add G'' to base graph, go to line 7
```

```
14.         else go to line 19
15.       if can identify a subgraph G'' with DOF=8 including base graph and weight of e is 3
16.         if {G'',e} is one case of rigid subgraphs of DOF =5
17.           add G'' to base graph, go to line 7
18.         else go to line 19
19.     an over-constrained subgraph {G'', e} is found ,break
20.     no over-constrained subgraph exists, add constraint e to G',
21.  endfor
```

Algorithm 2, for identifying a subgraph with DOF=n, is based on the "maximum flow" algorithm of Ford-Fulkerson [23]. For a given graph G', by distributing the weights of the constraints to the DOF of the entities, we can convert the structure graph G'{V, E} to a directed graph.

A new constraint network graph (s, V*, t, E*, w) is constructed for finding the direction of the arcs in a constraint graph G(V, E). The vertices in V* include the vertices in V and the arcs in E. s is the source, connected to every constraint in E. The capacity of the arc (s, v), $v \in E$ is equal to the weight of v in E. t is the sink connected to every node in V. The capacity of the arc (v, t), $v \in V$ is equal to the weight of v in V. The capacity of the network arc (v_1, v_2), $v_1 \in E$, $v_2 \in V$, v_2 being constrained by v_1, is infinite. The arcs in E* include the arcs between s and the vertices in V*, between t and the vertices in V*, and the arcs between the vertices in V*. w is the capacity of the arcs in E*. For example, Fig. 4(b) shows the associated network graph of the object in Fig. 4(a). The source can be viewed as a producer of some sort of fluid, and the sink as a consumer of the fluid. The maximum flow algorithm determines the largest possible amount of flow that can be sent from the source to the sink along each of the arcs in the constraint network graph. This maximum flow enables the conversion of the constraint graph into a directed graph via flow operations. In a network graph, for every arc (*e*, v), $e \in E$, $v \in V$, v is constrained by *e*. If the flow of *e* is not equal to zero, then the direction of *e* is directed to v in the constraint graph.

In a graph with DOF=n, after adding n extra constraints to the graph, the weight of each entity is equal to the sum of the weights of the constraints directed to it. All the weights of the constraints are distributed to the entities connected by the constraints. If the weight of one constraint is increased by 1 in a graph with DOF=n, after distributing all the constraints, the flow going out from the source is less than the weight of the constraint, i.e. one of the constraints cannot be totally absorbed by the entities.

Before adding the last constraint *e*, G' is under or well constrained. To identify a subgraph with DOF=n including *a, b*, which are the two nodes connected by *e, A*lgorithm 2 proceeds as follows: First, *n* extra constraints are added to the nodes *a, b*, where n is the DOF of the subgraph that we are finding. All the constraints in E', which consists of the constraints connecting the entities in the *base graph* and other entities in *G'*, are deleted. Then the constraints in E' are added one by one to G'. For every constraint in E', the weight of the constraint is increased by one to check the existence of a subgraph with DOF=n in G'. By distributing all the constraints, if one weight of a constraint cannot be absorbed by the entities, then a subgraph with DOF=n exists. Now by doing a breadth-first search, beginning from the last added constraint in E', we can identify a set of entities, in which the weight of every entity is equal to the sum of the weights of the constraints distributed to the entity. Then a subgraph G'' can be found and identified by the set of entities. *G''* is a subgraph with DOF=n if we add n extra constraints on nodes *a* and *b*, so DOF of *G''* is n.

When a new constraint is added in line 7 of Algorithm 2, the new maximum flow can be found more easily, because we only need to find the new argument path [23] for the newly added constraint.

The technique here is similar to the one used by Hoffmann [22], but Hoffmann distributed arcs connected to an entity, our method distributes arcs connect to the base graph. In 3D cases, when the entities are points, lines and planes, Hoffmann's method will stop at identifying every basic constraint types listed in Table 2 as dense subgraphs; a dense graph maybe well-constrained or over-constrained subgraph, so his method cannot identify a subgraph with certain DOF.

Algorithm 2: Identifying subgraph with DOF=n

Input (*G'*, *base graph*, *n*), where *n* is 5, 6, 7 or 8.

```
1.  int k=0
2.  set the weights of all the arcs in base graph to 0, and weights of all the nodes except for a, b to zero.
3.  add n extra constraints to the nodes a, b.
4.  delete all the constraints in E' which connect the entities in base graph and other entities in G'
5.  for every constraint e' in E'
```

```
6.         add e' to G'
7.         increase the weight of e' by 1
8.         use maximum flow method to distribute all the constraints to the entities. The constraints include constraints in
               G', n extra constraints and e'
9.         **if** one weight of a constraint cannot be distributed
10.          { k=k+1;  obtain a subgraph $G_k$'' }
11.        decrease weight of e' by 1.
12.    **endfor**
13.    **if** k=0 **return** G''
14.    **else return** no subgraph with DOF=n
```

The two theorems below prove that an over constrained subgraph {*G'', e*} is a minimal over-constrained subgraph in G, and it is correct.

Theorem 2 The over-constrained subgraph in Algorithm 1 contains the last added constraint e.
Proof
Let A be an over-constrained subgraph that does not contain the constraint e. Then there should be an arc that was in an over-constrained subgraph before e was considered. This contradicts the assumption that all arcs in G' have not been in an over-constrained subgraph. ■

Theorem 3 The over-constrained subgraph {*G'', e*} detected in Algorithm 1 is the minimal over-constrained subgraph.
Proof
If there is a smaller over-constrained subgraph G* inside *G''*, for all the constraints connecting the nodes between *G''* and G*, the directions of the constraints are from G* to *G''*, then it is impossible to find *G''* after G*, so {*G'', e*} is a minimal over- constrained subgraph.
Assume there is another minimal over-constrained subgraph which intersects the detected one. If the intersection contains more than {a, b}, then {*G'', e*} cannot be found as a minimal over-constrained subgraph. This can be proved by the method stated in the last paragraph.
If the intersection is {a, b}, Algorithm 2 can detect more than one subgraph G_k''. This special case is not dealt with in Algorithm 1, to keep it simple and easy to follow; we describe it here instead. In line 19 of Algorithm 1, if more than two of {*Gi'', e*} (i=0,1…k) are detected as over constrained subgraphs, the minimal over-constrained subgraph in Algorithm 1 is then set to {*e*} in this case. We will simply delete constraint *e* from the constraint graph to keep it from over-constrained. ■

In 2D, the entities are points and lines, and the weights of both of them are 2, and the weight of a constraint is 1; the DOF of all the rigid subgraphs in 2D are 3. So in the algorithm for finding a minimum over-constrained subgraph, for each e in *G'*, we need only to find if there exists a rigid subgraph with DOF=3 including the two end points of constraint e. If such a rigid subgraph exists, then *G''* is returned from Algorithm 2.

3. COMPLEXITY ANALYSIS

The complexity of Algorithm 2 is dominated by that of the Ford-Fulkerson algorithm, which is $O(mn\,(n+m/2))$, where n is the number of nodes and m is the number of constraints in a constraint graph. It can be shown that that on average**,** there are 4m/n constraints connected to a base graph, so the complexity of Algorithm 2 is $O(m^2\,(n+m/2))$. Then, the complexity of Algorithm 1 is $O(m^3\,(n+m/2))$.

4. EXAMPLE OF APPLICATION

In this section, we show three examples on how the algorithm of this paper can be used to identify the over-constrained subgraphs in a constraint graph.

4.1 A simple 2D example

In Fig. 4, the constraint graph consists of five 2D points and seven point-point distances identified by the lines between them.

According to Latham's method, if we add two extra constraints to point p_1, and one extra constraint to point p_2, shown in Fig. 4(c), then after the computation of maximal flow, there will not be any unsaturated constraint or unsaturated entity i.e. there will be no over-constraint and under-constraint in the example. But over and under constraint exist in the example: $p_2p_3p_4p_5$ is an over-constrained subgraph, and line p_1p_2 is under constrained, as it can rotate around p_2.
Our algorithm begins with G' which contains all the entities, then the constraints are added one by one to G'. If the distance constraint between p_2 and p_4 is the last constraint added, by distributing the DOF of the constraints to the DOF of the entities, a directed graph is obtained, as shown in Fig. 4(d). The minimum over-constrained subgraph $p_1p_2p_3p_4$ can be deduced from it, because the graph G'' $p_1p_2p_3p_4$ without the distance between p_2 and p_4 is a well-constrained subgraph with DOF=3, which is identified by Algorithm 2. We can make $p_1p_2p_3p_4$ well constrained by deleting any constraint inside it or the distance constraint between p_2 and p_4. After deleting one constraint, the point p_1 is still unsaturated, so the under constraint still exists in the constraint graph.

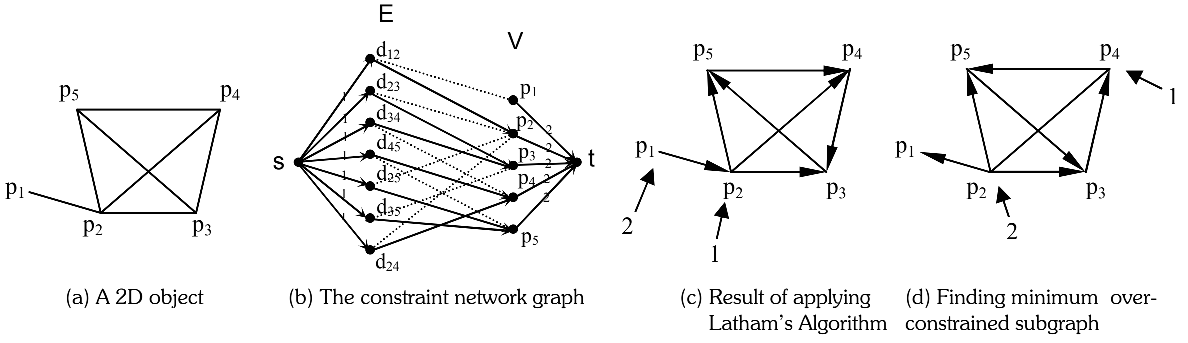

p5
p4
p1
p2
p3
(a) A 2D object
E
V
s
t
d12
d23
d34
d45
d25
d35
d24
(b) The constraint network graph
1
2
(c) Result of applying Latham's Algorithm
(d) Finding minimum over-constrained subgraph

Fig. 4. A 2D example on finding minimum over-constrained subgraph

4.2 A simple 3D example

Fig. 5 shows an example in 3D. The V of its constraint graph is composed of two planes f_1, f_2, two points p_1, p_2, and one line e_1. The E of the constraint graph is composed of parallel and distance constraints between f_1 and f_2, the angle between e_1 and f_1, the distance between p_1 and p_2, with p_1 lying on f_1, p_2 lying on f_2, p_1 lying on e_1 and p_2 lying on e_1.

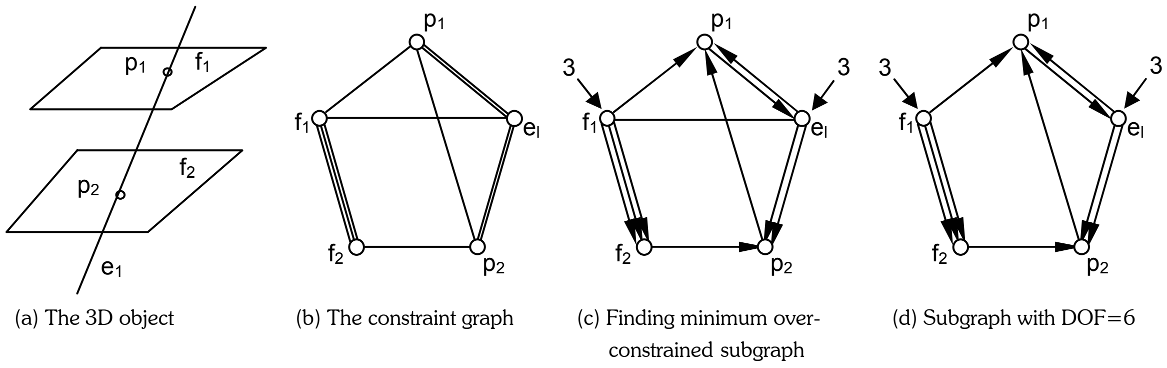
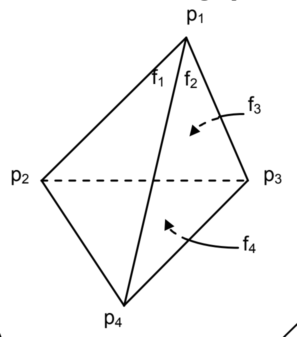


Fig. 5. A 3D example

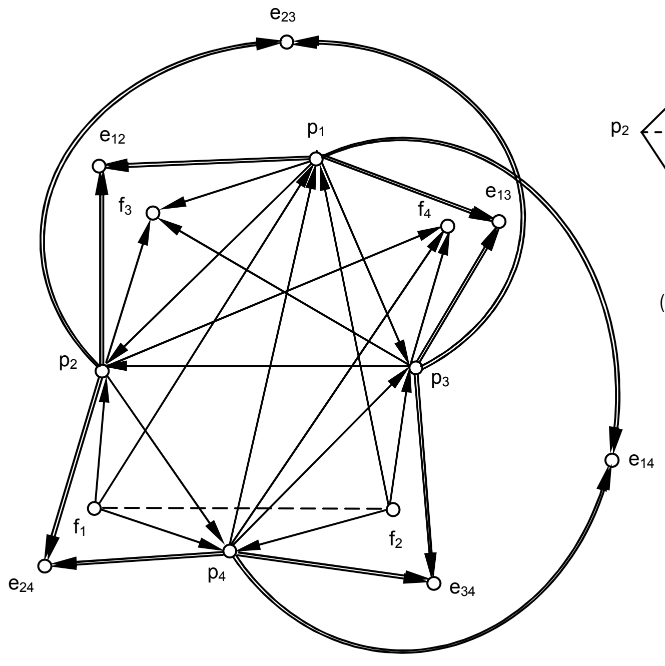
At the beginning, G' contains all the entities only, then the constraints are added to it one by one. If the angle constraint between e_1 and f_1 is the last constraint to be added, then Algorithm 1 proceeds as follows: firstly, identify rigid subgraphs with DOF=5 by calling Algorithm 2, which adds 5 extra constraints to e_1 and f_1. No rigid subgraph with DOF=5 is found. Next Algorithm 1 looks for subgraphs with DOF=6. Algorithm 2 similarly adds 6 extra constraints to e_1 and f_1, and a subgraph *G''*, $f_1f_2p_2e_1p_1$ is found. The DOF of the subgraph is 6, as shown in Fig. 5(d). Then, the minimum over-constrained subgraph is identified (see Fig. 5(c)), which includes $f_1f_2p_2e_1p_1$ and the constraint between f_1 and e_1. So, one of the constraints in the minimal over-constrained subgraph should be deleted. If we add the constraints into G' from the beginning in a different sequence, the same over-constrained subgraph will be identified too.

4.3 Another 3D example

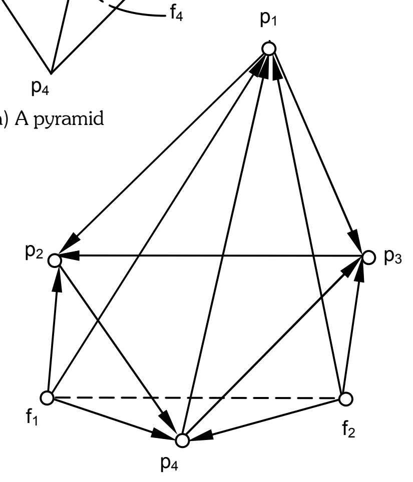
Fig. 6 gives another example using a 3D pyramid. This pyramid contains four points, six lines, four planes and six distance constraints with a weight of value one each, which are the distances between the points. In the pyramid, the four points are each constrained to lie on three planes, and therefore there are 12 constraints of point on plane, with a weight of one each. Similarly, there are 12 constraints of point on line with a weight of two each. If a user tries to add another constraint denoted by the dash line in Fig. 6(b), say the angle between two planes f_1 and f_2, the graph would be over-constrained, then a minimal over-rigid subgraph would be detected (see Fig. 6(c)). The minimal over-rigid subgraph includes all the points, the distances between them, f_1, f_2 and the angle constraint. Then we get a well-constrained system after deleting one constraint in the minimal over-constrained subgraph.



(a) A pyramid



(b) The pyramid's constraint graph



(c) The detected minimal over-constrained subgraph

Fig. 6. A 3D example using a pyramid

The examples in this section are necessarily simple, because otherwise the constraint diagrams become too complex and unwieldy to show here. Nevertheless, they illustrate the way the algorithms work.

5. CONCLUSION

This paper describes a flow-based method for identifying over-constrained subgraphs in a constraint system. Basically, it exploits the degree of freedom and degree of rigidity properties of subgraphs, and its method of adding the constraints one-by-one to the entities in the graph allows all the over-constrained subgraphs to be identified and corrected, as the adding progresses.

The algorithm has been implemented in our test system, and it works correctly. We are currently using the method to establish constraints automatically in 3D models reconstructed from 2D sketches.

Experiments show that the algorithm can identify over-constraint both in 2D and 3D, which correct the mistake of Latham's algorithm, and can handle 3D constraint problems where entities are points, lines and planes.

REFERENCES

[1] Sitharam, M., Combinatorial approaches to geometric constraint solving: problems, progress, directions, to appear in AMS-DIMACS book on computer aided design and manufacturing. Edited by Dutta, Janardhan, Smid. 2004.

[2] Lipson, H., Shpitalni, M., Optimization-based reconstruction of a 3D object from a single freehand line drawing, Computer Aided Design, Vol. 28, No. 8 1996, pp 651-663.

[3] Liu, J.Z., and Lee, Y.T., A graph-based method for face identification from a single 2D line drawing, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 23, No. 10, 2001, pp 1106–1119.

[4] Marill, T., Emulating the human interpretation of line drawings as three-dimensional objects, Computer Vision, Vol. 6, No. 2, 1991, pp 147-161.

[5] Langbein, F. C., Marshall, A. D., Martin, R. R., Choosing consistent constraints for beautification of reverse engineered geometric Models, Computer Aided Design, Vol. 36, No. 3, 2004, pp 261-278.

[6] Light, R., and Gossard, D. C., Modification of geometric models through variational geometry, Computer Aided Design, Vol. 14, No. 4, 1982, pp 209-214.

[7] Ait-Aoudia, S., Jegou, R., and Michelucci D., Reduction of constraint systems, In Compugraphics, Alvor, Portugal, 1993, pp 83-92.

[8] Buchanan, S.A. and Pennington, A., Constraint definition system: a computer-algebra based approach to solving geometric-constraint problems, Computer Aided Design, Vol. 25, No. 12, 1998, pp 741-750.

[9] Gao, X.S. and Chou, S.C., Solving geometric constraint systems. II, A symbolic approach and decision of Rc-constructibility. Computer Aided Design, Vol. 30, No. 2, 1998, pp 115-122.

[10] Owen, J., Algebraic solution for geometry from dimensional constraints, ACM Symposium on Foundations of Solid Modeling, Austin, TX, 1991, pp 397–407.

[11] Bouma, W., Fudos, I., Hoffmann, C.M., Cai, J., and Paige, R., A Geometric constraint solver, Computer Aided Design, Vol. 27, No. 6, 1995, pp 487-501.

[12] Joan-Arinyo, R., Soto-Riera, A., Vila-Marta, S., Vilaplana-Pastó, J., Transforming an under-constrained geometric constraint problem into a well-constrained one, Proceedings of the eighth ACM symposium on Solid modeling and application, Seattle, Washington, USA, June 2003, pp 33-44.

[13] Fudos, I., Constraint Solving for Computer Aided Design, PhD Thesis, Purdue University, West Lafayette, IN, USA, 1998.

[14] Fudos, I. and Hoffmann, C.M., Correctness proof of a geometric constraint solver, International Journal of Computational Geometry & Applications, Vol. 6, No. 4, 1996, pp 405-420.

[15] Fudos, I. and Hoffmann, C.M., A graph-constructive approach to solving systems of geometric constraints, ACM Transactions on Graphics, Vol. 16, No. 2, 1997, pp 179-216.

[16] Latham R. S., and Middleditch, A. E., Connectivity analysis: a tool for processing geometric constraints, Computer Aided Design, Vol. 28, No. 11, 1996, pp 917-928.

[17] Hoffmann, C.M., Lomonosov, A., Sitharam, M., Decomposition plans for geometric constraint systems, part I: performance measures for CAD, Journal of Symbolic Computation, Vol. 31, No. 4, 2001, pp 367-408.

[18] 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, Vol. 34, No. 2, 2002, pp 97-108.

[19] Kramer, G., Solving Geometric Constraint Systems, MIT Press, Cambridge, MA, USA, 1992.

[20] Jermann, C., Neveu, B., Trombettoni, G., A new structural rigidity for geometric constraint systems, In proceedings of the 4th International Workshop on Automated Deduction in Geometry, Hagenberg Castle, Austria, September 4-6, 2002, pp 87-106.

[21] Jermann, C., Neveu, B., Trombettoni, G., Algorithms for identifying rigid subsystems in geometric constraint systems, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003, pp 233-238.

[22] Hoffmann, C.M., Lomonosov, A., Sitharam, M., In: Smolka G, editor, Finding solvable subsets of constraint graphs, LNCS 1330, Springer, New York, NY, 1997, pp 463–477.

[23] Gondran, M., Graphs and Algorithms, Wiley, New York, NY, USA, 1982.