A Tractable, Approximate, Combinatorial 3D rigidity characterization

Meera Sitharam [*][†] Yong Zhou[*]

June 20, 2004

Abstract

There is no known, tractable, characterization of 3D rigidity of sets of points constrained by pairwise distances or *3D distance constraint graphs*. We give a combinatorial *approximate* characterization of such graphs which we call *module-rigidity*, which can be determined by a polynomial time algorithm. We show that this property is natural and robust in a formal sense. Rigidity implies module-rigidity, and module-rigidity significantly improves upon the generalized Laman degree-of-freedom or density count. Specifically, graphs containing "bananas" or "hinges" [8] are not module-rigid, while the generalized Laman count would claim rigidity. The algorithm that follows from our characterization of module-rigidity gives a complete decomposition of non module-rigid graphs into its maximal module-rigid subgraphs.

To put the result in perspective, it should be noted that, prior to the recent algorithm of [21] there was no known polynomial time algorithm for obtaining all maximal subgraphs of an input constraint graph that satisfy the generalized Laman count, specifically when overconstraints or redundant constraints are present.

The new method has been implemented in the FRONTIER [23], [35], [28], [29] opensource 3D geometric constraint solver and has many useful properties and practical applications [30], [31], [32], [34], [33]. Specifically, the method is used for constructing a so-called decomposition-recombination (DR) plan for 3D geometric constraint systems, which is crucial to defeat the exponential complexity of solving the (sparse) polynomial system obtained from the entire geometric constraint system. The DR-plan guides the algebraic-numeric solver by ensuring that only small subsystems are ever solved. The new, approximate characterization of 3D rigidity permits FRONTIER to deal with a far larger class of 3D constraint systems (a class adequate for most applications) than any other current geometric constraint solver.

Keywords: Combinatorial Rigidity, Variational geometric constraint solving, Cyclical and 3D geometric constraint systems, Decomposition of geometric constraint systems, Underconstrained and Overconstrained systems, Degree of Freedom analysis, Constraint graphs.

1 Introduction

A *3D distance constraint graph* is a weighted graph with vertices representing point objects in 3D and edges representing distance constraints between the points. The weight of each vertex is 3, representing its 3 positional *degrees of freedom (dof)*, and the weight of each edge is 1, representing the number of degrees of freedom the constraint removes.

The constraints can be written as quadratic equations in variables representing the coordinates of the points For example, a distance constraint of d between two points (x_1, y_1, z_1) and (x_2, y_2, z_2) in 3D is written as $(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_1 - z_2)^2 = d^2$. The resulting *3D distance constraint system* is said to be *generically rigid*, if it has at most finitely many incongruent solutions (i.e., its solution set, the corresponding algebraic variety, is finite modulo rotations and translations) in the generic case (i.e., when a generic, algebraically independent set of values is chosen for the distance constraints).

Thus the property of generic rigidity of a distance constraint system - being independent of the actual distance values - is in fact a property of the underlying distance constraint graph alone. (We call the corresponding constraint graph *rigid*). One would expect a purely combinatorial characterization (and corresponding algorithm) for determining rigidity of distance constraint graphs. While Laman's theorem [19] gives such a characterization for 2D distance constraint graphs, no such characterization has been proven for 3D, although several conjectures

[*]University of Florida; Work supported in part by NSF Grant CCR 99-02025, NSF Grant EIA 00-96104
[†]*corresponding author: sitharam@cise.ufl.edu*

exist [8]. A (real) solution or embedding or *realization* of a distance constraint system is sometimes called a *framework*. There is a characterization of rigidity of a distance constraint graph using so called *infinitesimal rigidity* of frameworks and the associated rigidity matroids [8]. This characterization asserts full generic rank of a so-called rigidity matrix (its entries are vectors determined by the coordinate positions of the constrained pairs of points, expressed as indeterminates). However, this characterization does not yield a polynomial time algorithm for determining rigidity of a distance constraint graph. In fact, none of the combinatorial characterization conjectures appears to translate to a polynomial time algorithm.

Here, we adopt a different tack. We give a combinatorial *approximate* characterization of 3D rigidity, which we call *module-rigidity*, which can be determined by a polynomial time algorithm. We show that this property is natural and robust in a formal sense. Rigidity implies module-rigidity, and module-rigidity significantly improves upon the generalized Laman degree-of-freedom or density count. Specifically, graphs containing "bananas" or "hinges" [8] are not module-rigid, while the generalized Laman count would claim rigidity. More precisely:

$$\text{rigid} \subseteq \text{module-rigid} \subset \text{generalized Laman or dof rigid (contains bananas and hinges)}$$

The algorithm that follows from our characterization of module-rigidity has a number of useful properties. Many of these are based on the fact that the algorithm gives a complete decomposition of non module-rigid graphs into its maximal module-rigid subgraphs. The new method has been implemented in the FRONTIER [23], [35], [28], [29] opensource 3D geometric constraint solver (2003 and 2004 versions) and has many practical applications: geometric constraint systems are used as succinct, minimal, editable representations of geometric composites in many contexts including mechanical computer aided design, robotics, molecular modeling and teaching geometry For recent reviews of the extensive literature on geometric constraint solving see, e.g, [14, 18, 6]. Most of the geometric constraint solvers so far deal with 2D constraint systems, although some of the newer approaches including [11, 12, 16, 17] [15, 2, 24] [10, 21, 23, 35], extend to 3D constraint systems. These constraint solvers have been compared with respect to various formal performance measures in [16]. The new, approximate characterization of 3D rigidity permits FRONTIER to deal with a far larger class of 3D constraint systems (a class adequate for most applications) than any other current constraint solver.

Most geometric constraint solvers rely on recursively decomposing the input constraint system into small, generically rigid subsystems prior to solving. These subsystems are fed to an off-the-shelf algebraic-numeric solver, and the solutions are recombined to generate a solution to the entire constraint system. This decomposition-recombination (DR) plan (defined in Section 1.1) is crucial to defeat the exponential complexity of solving the (sparse) polynomial system obtained from the entire geometric constraint system, by guiding the algebraic-numeric solver effectively: only small subsystems are ever solved. It is also crucial that the DR-plan be generated efficiently, certainly in polynomial time. As a result, DR-planners are usually graph algorithms that combinatorially determine rigidity of the subsystems in the DR-plan.

For the DR-plan to effectively guide the solver, the subsystems in the DR-plan need to be rigid so that their solutions can be recombined. However, if the DR-planner should err occassionally by falsely claiming rigidity, this error will be detected during the actual solving process.

In Section 1.1 we give the basic background on geometric constraint graphs, generic rigidity, generalized Laman or dof analysis, DR-plans and their basic properties. The characterization of module-rigidity (and the corresponding algorithm) presented in Section 3 build upon an alternate characterization (and corresponding Frontier vertex algorithm) of [21] of generalized Laman or dof rigidity for 3D geometric constraint graphs that is based on DR-plans. This characterization has many useful properties [21], [31], [32], [30], [34], [33] which are inherited by the new module-rigidity characterization, specifically because it gives a polynomial time algorithm for determining a complete decomposition into maximal dof rigid subgraphs, if the graph is not dof rigid. Most importantly, the algorithm works in the presence of redundant or overconstraints. Earlier graph algorithms such as [9] for determining dof rigidity rely on the removal of overconstraints. While [10] provide a method for removing overconstraints without making the whole graph dof underconstrained, this could make dof rigid subgraphs underconstrained. Hence methods such as [9] do not provide complete decompositions.

1.1 Constraint Graphs, Degrees of Freedom, DR-Plans

Geometric constraint graphs are a generalization of the distance constraint graphs to which the new results of this paper are restricted. However, the concepts in this and the next sections apply to general constraint graphs as well. A *geometric constraint graph* $G = (V, E, w)$ is a weighted graph with n vertices (representing geometric objects) V and m edges (representing constraints) E; $w(v)$ is the weight of vertex v and $w(e)$ is the weight of edge e, corresponding to the number of degrees of freedom available to an object represented by v and number of degrees of freedom *(dofs)* removed by a constraint represented by e respectively.

Several 3D distance constraint graphs whose vertices of weight 3 (representing points) and edges of weight 1 (representing distance) can be found in Figures 1, 2, 3, 11, 10.

A subgraph $A \subseteq G$ that satisfies

$$\sum_{e \in A} w(e) + D \geq \sum_{v \in A} w(v) \tag{1}$$

is called *dense*, where D is a dimension-dependent constant, to be described below. Function $d(A) = \sum_{e \in A} w(e) - \sum_{v \in A} w(v)$ is called *density* of a graph A. Its magnitude is also called the *generalized Laman* or *dof count*, since it is a natural generalization of Laman's theorem [19] that gives a combinatorial characterization of rigidity for 2D distance constraint graphs.

The constant D is typically $\binom{d+1}{2}$ where d is the dimension. The constant D captures the degrees of freedom of a rigid body in d dimensions. For 2D contexts and Euclidean geometry, we expect $D = 3$ and for 3D contexts $D = 6$, in general. If we expect the rigid body to be fixed with respect to a global coordinate system, then $D = 0$.

Next, we give some purely combinatorial properties of constraint graphs based on density. These will be later shown to be related to properties of the corresponding constraint systems.

A dense graph with density strictly greater than $-D$ is called *overconstrained*. A graph that is dense and all of whose subgraphs (including itself) have density at most $-D$ is called *dof wellconstrained*. A graph G is called *dof well-overconstrained* if it satisfies the following: G is dense, G has atleast one dof overconstrained subgraph, and has the property that on replacing all dof overconstrained subgraphs by dof wellconstrained subgraphs (in any manner), G remains dense. A graph that is dof wellconstrained or dof well-overconstrained is said to be *dof rigid* or a *dof cluster*. A dense graph is *minimal* if it has no dense proper subgraph. Note that all minimal dense subgraphs are dof clusters but the converse is not the case. A graph that is not a dof cluster is said to be *dof underconstrained*. If a dense graph is not minimal, it could in fact be an dof underconstrained graph: the density of the graph could be the result of embedding a subgraph of density greater than $-D$. A *trivial* graph is any graph that reduces (by resolving incidences) to a single point in 2D or to a fixed or variable length line segment in 3D. All these trivial graphs have rotational symmetries and are dof rigid; the former two cases are truly rigid and are dof overconstrained.

To discuss how the graph theoretic properties based on *degree of freedom (dof) analysis* described above relate to corresponding properties of the corresponding constraint *system*, we need to introduce the notion of *genericity*. Formally we use the notion of genericity of e.g, [3]. A property is said to hold *generically* for polynomials $f_1, \dots, f_n$ if there is a nonzero polynomial P in the coefficients of the f_i such that this property holds for all $f_1, \dots, f_n$ for which P does not vanish.

Thus the constraint system E is generically rigid if there is a nonzero polynomial P in the coefficients of the equations of E - or the parameters of the constraint system - such that E has at most finitely many zeroes modulo rotations and translations, when P does not vanish. For example, if E consists of distance constraints, the parameters are the distances. Even if E has no overt parameters, i.e, if E is made up of constraints such as incidences or tangencies or perpendicularity or parallelism, E in fact has hidden parameters capturing the extent of incidence, tangency, etc., which we consider to be the parameters of E. (Generically overconstrained systems have no zeroes when P does not vanish and generically underconstrained systems have infinitely many zeroes, i.e., a non-zero-dimensional algebraic variety; both generically wellconstrained and generically overconstrained systems are said to be generically rigid; the latter are sometimes refered to as redundantly rigid).

1.2 Inadequacy of a generalized Laman or dof analysis

A generically rigid system always gives a dof cluster, but the converse is not always the case. In fact, there are dof well-constrained clusters whose corresponding systems are not generically rigid and are in fact generically not rigid. The root cause of these misclassifications is the presence of “hidden” dependent constraints that cannot be found by a dof count.

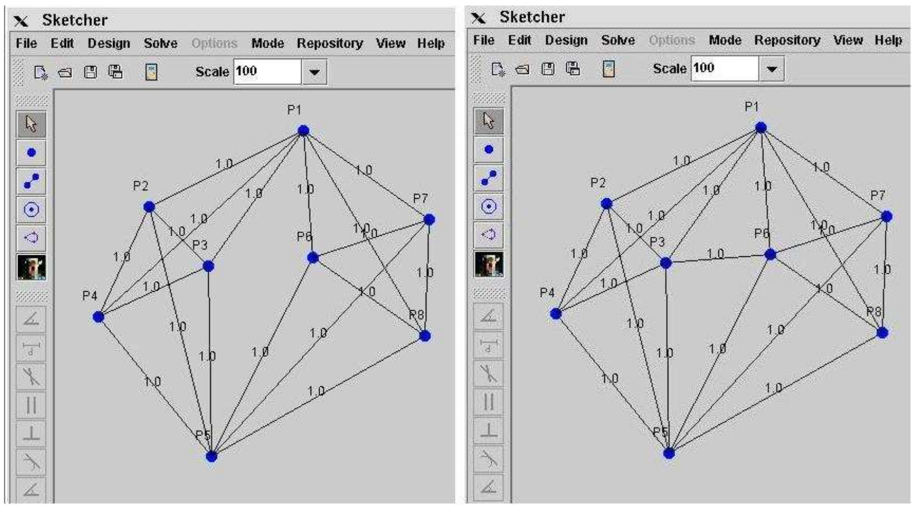

Sketcher
File Edit Design Solve Options Mode Repository View Help
Scale 100
P1
P2
P3
P4
P5
P6
P7
P8
1.0
Sketcher
File Edit Design Solve Options Mode Repository View Help
Scale 100
P1
P2
P3
P4
P5
P6
P7
P8
1.0

Figure 1: 3D constraint system drawn on a 2D canvas ilustrating the bananas problem; corresponding constraint graphs have vertices of weight 3 and edges of weight 1; see text for explanation

Consider for example the Figures 2, 1, 3, related to the so-called "bananas" problem in 3D, which is a type of constraint dependence occurring in a large class of 3D distance constraint graphs, although this detection is nontrivial.

A dof analysis of the 3D constraint system in Figure 2(top) would report the left and right subsystems ($P1, P2, P3, P4, P5$ and P_1, P_6, P_7, P_8, P_5 respectively) and the whole system to be dof wellconstrained clusters. Figure 3 (bottom) has the same number of constraints, but a dof analysis would report both that the left subgraph as dof overconstrained and the whole as dof underconstrained. Figure 1 (left) also has the same number of constraints and is a dof rigid cluster. However, while the left and right subsystems are (in fact) generically rigid, the whole system is generically overconstrained. In a well-defined sense, this caused by a constraint dependence. On the other hand, when *restricted to consistently overconstrained situations* (those choices of distances - such as in this example - that are guaranteed to admit a solution), the system in Figure 1 (left) is generically underconstrained, although the system on Figure 1(right) is generically wellconstrained.

In fact, a constraint system is generically overconstained if the common overlap of *any* subset of its dof wellconstrained clusters is dof underconstrained. The above "bananas" is a special case of this. However, the dof analysis is inaccurate only in the "bananas" situation. Another standard example, in 4 dimensions the graph $K_{7,6}$ representing distances is minimal dense, and hence a dof rigid cluster, but it does not represent a generically rigid system.

However, as mentioned earlier, in 2 dimensions, according to Laman's theorem [19] if all geometric objects are points and all constraints are distance constraints between these points then any minimal dense dof rigid cluster represents a generically rigid system.

In fact, there is no known, tractable characterization of generic rigidity of distance systems for 3 or higher dimensions, based purely properties of the constraint graph. In fact even in 2D, while Laman's theorem [19] combinatorially characterizes generic rigidity of point and distance systems, there are no known combinatorial characterizations of rigidity, when other constraints besides distances are involved.

For example, in the case of angle constraints in 2D: 3 line segments with 3 incidence constraints form a triangle with 3*4-3*2 = 6 (resp. 3*6-3*3 = 9) degrees of freedom. It would appear that to make it wellconstrained, we can introduce 3 angle constraints (each of which removes 1 dof). But in fact, this would make it generically overconstrained.

1.3 The need for decomposition: dof DR-plans and their properties

The overwhelming cost of solving a geometric constraint system is the size of the largest subsystem that is solved using a direct algebraic/numeric solver. This size dictates the practical utility of the overall constraint solver, since the time complexity of the constraint solver is at least *exponential* in the size of the largest such

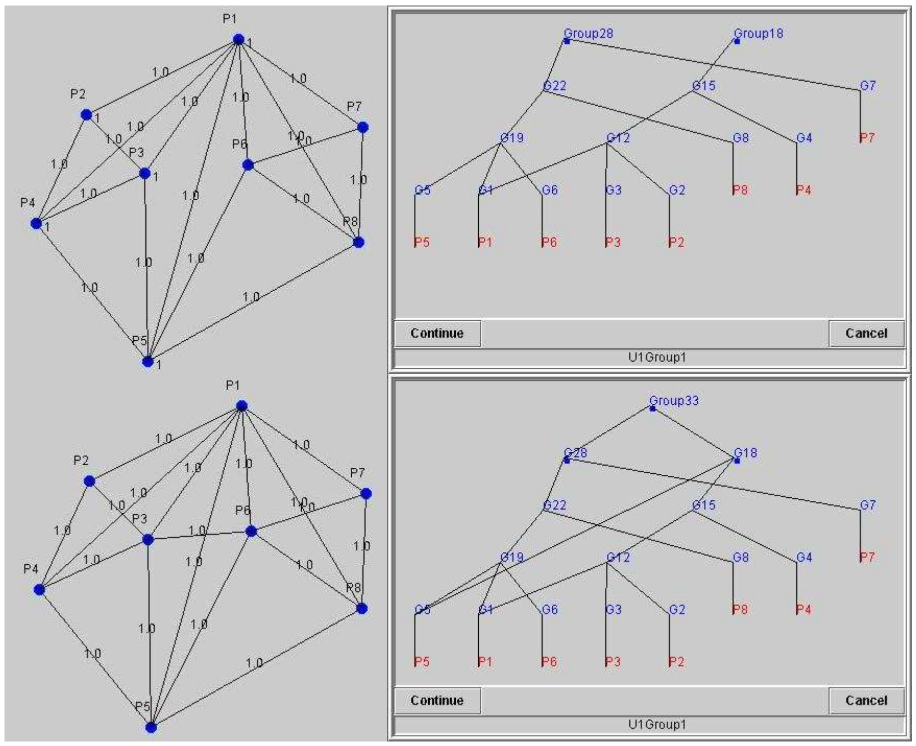


Figure 2: Modifications to 3D system in Figure 1: dof wellconstrained (DR-plan has single source, top) and underconstrained (DR-plan has many sources, bottom); see text for explanation

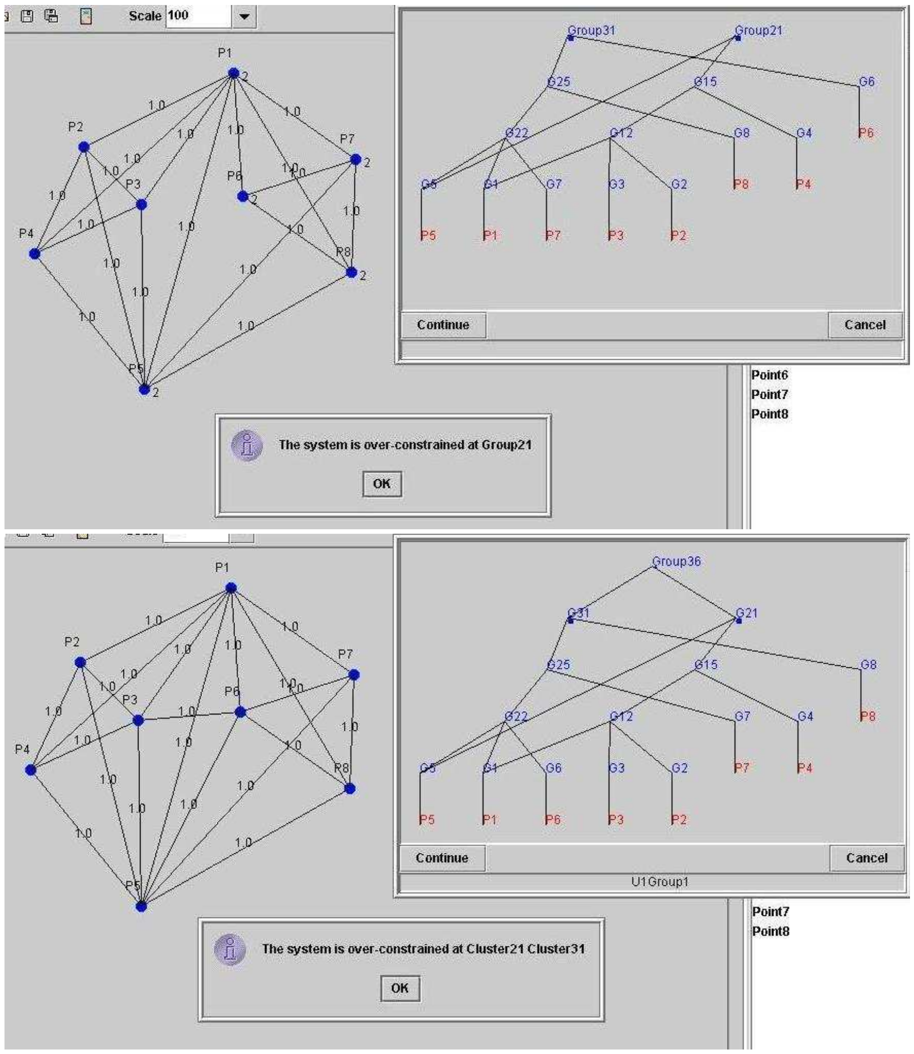
subsystem.

The DR-planner is a graph algorithm that outputs a *decomposition-recombination plan (DR-plan)* of the constraint graph. In the process of combinatorially constructing the DR-plan in a bottom up manner, at stage i, it locates a dof rigid subgraph S_i in the current constraint graph G_i, and uses an abstract *simplification* of S_i to to create a transformed constraint graph G_{i+1}.

Decomposition algorithms based on constraint graphs have been proposed since the early 90's based on recognition of subgraph patterns such as triangles [7, 25, 26, 24] [20, 22]; and based on Maximum Matching [27, 1]. However, prior to [16], the DR-planning problem and appropriate performance measures for the planners were not formally defined. That paper also gives a table comparing 3 main types of DR-planners, with respect to these performance measures. A subsequent paper [17] presents the framework of a DR-planner based on generalized dof analysis (beyond detecting specific patterns) that would optimize these performance measures. The complete dof-based DR-planner, called the Frontier vertex DR-planner, based on this framework, along with properties, proofs and applications is presented in [21]. These are sketched in Section 2 and form the starting point of the new characterization of module-rigidity and the corresponding algorithm presented in Section 3.

Formally, a *DR-plan* of a constraint graph G is a directed acyclic graph (DAG) whose nodes represent rigid subgraphs in G, and edges represent containment. The leaves or sinks of the DAG are all the vertices (primitive clusters) of G. The roots or sources are all the maximal rigid clusters of G. In a *partial* DR-plan, the last condition may not hold. There could be many DR-plans for G. See Figure 4. Note that the definition of DR-plans is robust (has a type of Church-Rosser property) in that any partial DR-plan can be extended to obtain a DR-plan for G. I.e., if the DR-plan is being built bottom up, any construction path will lead to a valid DR-plan. A *dof DR-plan* is one where each of the clusters is only required to be dof rigid, and the roots are required to be all the maximal dof rigid clusters of G. One can define a partial dof DR-plan analogously: the dof DR-plan definition is also robust in that any partial dof DR-plan extends to a dof DR-plan. An *optimal* (dof) DR-plan is one that minimizes the maximum fan-in. The *size* of a (dof) rigid cluster in a (dof) DR-plan is its fan-in (it represents the size of the corresponding subsystem, once its child clusters are solved).

All properties defined for DR-plans transfer as performance measures of the *DR-planners* or DR-planning



The system is over-constrained at Group21

The system is over-constrained at Cluster21 Cluster31

Figure 3: dof overconstrained clusters in well (single source in DR-plan) and underconstrained (multiple sources in DR-plan) graphs; see text for explanation

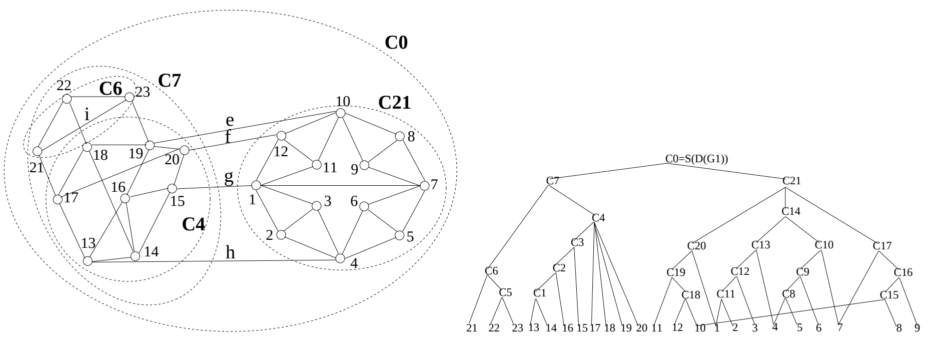


Figure 4: 2D distance constraint graph G1 and DR-plan; all vertices represent points and have weight 2 and edges represent distances and have weight 1

algorithms. It is shown in [21], that the problem of finding the optimal DR-plan of a constraint graph is NP-hard, and approximability results are shown only in special cases. Nonapproximability results are not known. This is the case even when one is only interested in a dof DR-plan. However, most DR-planners make adhoc choices during computation (say the order in which vertices are considered) and we can ask how well (close to optimal) the *best* computation path of such a DR-planner would perform (on the worst case input). We call this the *best-choice approximation factor* of the DR-planner.

As we shall see in the next section, a good (dof) DR-plan is crucial not only for solving efficiency, but for determining (dof) rigidity of the input constraint graph, as well as for underconstraint detection and completion [21], [32], is indispensable for navigation of the solution space, [31], [34], for dealing with overconstraints, [10] and for efficiently updating the constraint system [32]. All of these properties can be found in [29], [28]. A few other properties of DR-plans are of interest. We would like the *width* i.e, *number* of clusters in the DR-plan to be small, preferably at most cubic in the size of G: this reflects the complexity of the DR-planner.

2 Determining dof rigidity via complete, maximal decompositions: The Frontier Vertex Algorithm (FA) DR-Planner

In this section, we first give an alternative characterization of dof rigidity which translates to a useful property of dof DR-plans called *dof completeness.* (We omit proofs). Then we sketch relevant properties of Frontier vertex DR-plans and the corresponding DR-planner (FA DR-planner) [17, 21] which follows this characterization.

Let C be a geometric constraint graph. Then $Q = \{C_1, \dots, C_m\}$, a set of dof rigid proper subgraphs of C, is a *complete, maximal, dof rigid decomposition* of C if the following hold.

- If there is a maximal, dof rigid proper subgraph of C then it must contain one of the C_i in Q.
- Furthermore, Q should satisfy one of the following.
  Case 1: $m = 2$ and C_1 and C_2 intersect on a nontrivial subgraph and their union induces all of C
  Case 2: Each of the C_i's is *nearly maximal with respect to the set* Q in the following sense: the only dof rigid proper subgraphs of C that strictly contain C_i intersect all the other subgraphs C_j, $j \neq i$ on nontrivial subgraphs;

The next theorem gives an alternate characterization of dof rigidity.

Theorem 2.1 *Let C be a geometric constraint graph and $Q = \{C_1, \dots, C_m\}$, be a complete, maximal, dof rigid decomposition of C. Then C is dof rigid if and only if*

$$\sum_{S \subseteq Q} (-1)^{|S|-1} Adj - dof(\bigcap_{C_i \in S} (C_i)) \leq D,$$

where (recall) D is the number of dofs of a rigid body, and Adj-dof(C_i) is either the number of dofs (negation of density) of C_i if C_i is trivial; or simply D if C_i is nontrivial. Note that if Case 1 holds, then C is automatically dof rigid - in fact, the first property of Q is redundant.

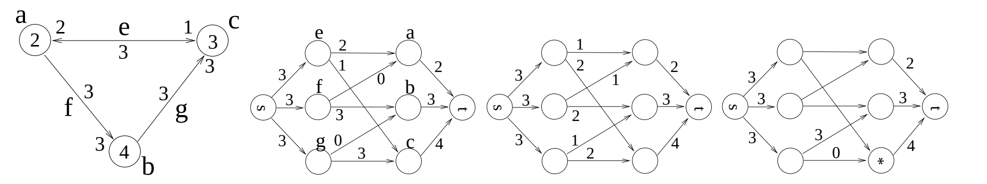


Figure 5: From Left. Constraint graph G with edge weight distribution. D is assumed to be 0 (system fixed in coordinate system); A corresponding flow in bipartite G^*. Another possible flow. Initial flow assignment that requires redistribution

The next lemma explains the tractability of this method of determining dof rigidity.

Lemma 2.2 *If C is not dof rigid, then only Case 2 in the Definition 2 applies. Furthermore, Case 2 implies that no pair of C_i intersect on more than a trivial subgraph. Thus (using a simple Ramsey theoretic argument), m is at most $O(n^3)$, where n is the number of vertices C. Furthermore, the computation of the inclusion-exclusion formula in Lemma 2.1 takes $O(n^3)$ time.*

This leads to a robust property of DR-plans using which the characterization can be translated to an algorithm.

A dof DR-plan P for a geometric constraint graph G is *dof complete* if the set Q of child clusters of every dof cluster C in P is a complete, maximal, dof rigid decomposition of C. Partial dof complete DR-plans are defined analogously as in Section 1.1, and just as before, any partial dof-complete) DR-plan for a constraint graph G can be extended to a dof-complete) DR-plan for G

2.1 The Frontier Vertex DR-plan (FA DR-plan)

Note. Throughout this section, unless otherwise mentioned, "cluster" means "dof cluster,'" "rigid" means "dof rigid," and "DR-plan" means "dof DR-plan."

Intuitively, an FA DR-plan is built by following two steps repeatedly:

1. *Isolate* a cluster C in the current graph G_i (which is also called the *cluster graph* or *flow graph* for reasons that will be clear below). Check and ensure a complete, maximal, dof rigid decomposition of C.

2. *Simplify* C into $T(C)$, transforming G_i into the next cluster graph $G_{i+1} = T(G_i)$ (the recombination step).

2.1.1 Isolating Clusters

The isolation algorithm, first given in [13, 14] is a modified incremental network maximum flow algorithm. The key routine is the *distribution* of an edge (see the DR-planner pseudocode in the Appendix of Part II) in the constraint graph G. For each edge, we try to *distribute* the weight $w(e) + D + 1$ to one or both of its endpoints as *flow* without exceeding their weights, referred to as "distributing the edge e." See *DistributeEdge* in the pseudocode in Part II, Appendix. This is best illustrated on a corresponding bipartite graph G^*: vertices in one of its parts represent edges in G and vertices in the second part represent vertices in G; edges in G^* represent incidence in G. As illustrated by Figure 5, we may need to redistribute (find an augmenting path).

If we are able to distribute all edges, then the graph is not dense. If no dense subgraph exists, then the flow based algorithm will terminate in $O(n(m + n))$ steps and announce this fact. If there is a dense subgraph, then there is an edge whose weight plus $D + 1$ cannot be distributed (edges are distributed in some order, for example by considering vertices in some order and distributing all edges connecting a new vertex to all the vertices considered so far). It can be shown that the search for the augmenting path while distributing this edge marks the required dense graph. It can also be shown that if the found subgraph is not overconstrained, then it is in fact minimal. If it is overconstrained, [13, 14] give an efficient algorithm to find a minimal (non-trivial, if one exists) dof cluster inside it. Then [21] gives a method to ensure a complete, maximal, dof rigid decomposition of C.

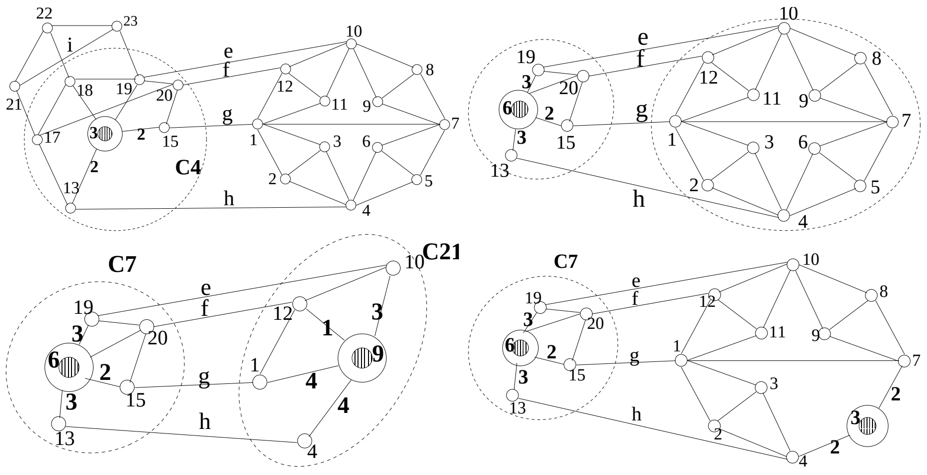


Figure 6: From left: FA's simplification of graph givin DR-plan in Figure 4; clusters are simplified in their numbered order: C4 is simplified before C7 etc.

2.1.2 Cluster Simplification

This simplification was given in [17, 15]. The found cluster C interacts with the rest of the constraint graph through its *frontier vertices*; i.e., the vertices of the cluster that are adjacent to vertices not in the cluster. The vertices of C that are not frontier, called the *internal vertices*, are contracted into a single *core* vertex. This core is connected to each frontier vertex v of the simplified cluster $T(C)$ by an edge whose weight is the the sum of the weights of the original edges connecting internal vertices to v. Here, the weights of the frontier vertices and of the edges connecting them remain unchanged. The weight of the core vertex is chosen so that the density of the simplified cluster is $-D$, where D is the geometry-dependent constant. This is important for proving many properties of the FA DR-plan: even if C is overconstrained, $T(C)$'s overall weight is that of a wellconstrained graph, (unless C is rotationally symmetric and trivial, in which case, it retains its dof or weight). Technically, $T(C)$ may not be wellconstrained in the precise sense: it may contain an overconstrained subgraph consisting only of frontier vertices and edges, but its overall dof count is that of a wellconstrained graph.

Figure 6 illustrates this iterative simplification process ending in the final DR-plan of Figure 4.

2.2 The Frontier Vertex Algorithm (FA DR-planner)

The challenge met by FA is that it provably meets several competing requirements. Specifically, it gives a dof complete DR-plan. The graph transformation performed by the FA cluster simplification is described formally in [17, 15] that provide the vocabulary for proving certain properties of FA that follow directly from this simplification. However, other properties of FA require details of the actual DR-planner that ensures them, and are briefly sketched here.

Note: a detailed pseudocode of the FA DR-planner (the existing version, as well as incorporating the module-rigidty algorithm of this paper) can be found in [35], [28]. The pseudocode has been implemented as part of the downloadable, opensource FRONTIER geometric constraint solver [23], [35], [28], [29] .

The basic FA algorithm is based on an extension of the *distribute* routine for edges (explained above) to vertices and clusters in order for the isolation algorithm to work at an arbitrary stage of the planning process, i.e, in the cluster or flow graph G_i.

First, we briefly describe this basic algorithm. Next, we sketch the parts of the algorithm that ensure 3 crucial, inter-related properties of the output DR-plan:
(a) ensuring dof completeness;
(b) for underconstrained graphs: outputing a complete set of maximal clusters as sources of the DR-plan;
(c) controlling width of the DR-plan to ensure a polynomial time algorithm.

Three datastructures are maintained. The current *flow or cluster graph*, G_i the current *DR-plan* (this information is stored entirely in the hierarchical structure of clusters at the top level of the DR-plan), and a *cluster queue*, which is the top-level clusters of the DR-plan that have not been distributed so far, in the order that they were found (see below for an explanation of how clusters are distributed). We start with the original graph (which serves as the cluster or flow graph initially, where the clusters are singleton vertices). The DR-plan consists of the leaf or sink nodes which are all the vertices. The cluster queue consists of all the vertices in an arbitrary order.

The method *DistributeVertex* (see pseudocode of Part II, Appendix) distributes all edges (calls DistributeEdge) connecting the current vertex to all the vertices considered so far. When one of the edges cannot be distributed and a minimal dense cluster C is discovered, its simplification $T(C)$ (described above) transforms the flow graph. The flows on the internal edges and the core vertex are inherited from the old flows on the internal edges and internal vertices. Notice that undistributed weights on the internal edges simply disappear. The undistributed weights on the frontier edges are distributed (within the cluster) as well as possible. However, undistributed weights on the frontier edges (edges between frontier vertices) may still remain if the frontier portion of the cluster is severely overconstrained. These have to be dealt with carefully. (See discussion on dealing with the problems caused by overconstraints below.) The new cluster is introduced into the DR-plan and the cluster queue.

Now we describe the method *DistributeCluster* Assume all the vertices in the cluster queue have been distributed (either they were included in a higher level cluster in the DR-plan, or they failed to cause the formation of a cluster and continue to be a top level node of the DR-plan, but have disappeared from the cluster queue). Assume further that the DR-plan is not complete, i.e., its top level clusters are not maximal. The next level of clusters are found by distributing the clusters curently in the cluster queue. This is done by filling up the "holes" or the available degrees of freedom of a cluster C being distributed by D units of flow. The *PushOutSide* method successively considers each edge incident on the cluster with 1 endpoint outside the cluster. It distributes any undistributed weight on these edges + 1 extra weight unit on each of these edges. It can be shown that if C is contained inside a larger cluster, then atleast one such cluster will be found by this method once all the clusters currently in the cluster queue have been distributed. The new cluster found is simplified to give a new flow graph, and gets added in the cluster queue, and the DR-plan as described above.

Eventually, when the cluster queue is empty, i.e, all found clusters have been distributed, the DR-plan's top level clusters are guaranteed to be the complete set of maximal dof rigid subgraphs of the input constraint graph. See [21] for formal proofs.

Note: Throughout, in the interest of formal clarity, we leave out ad hoc, but highly effective heuristics that find simple clusters by avoiding full-fledged flow. One such example is called "sequential extensions" which automatically creates a larger cluster containing a cluster C and a vertex v provided there are atleast D edges between C and v. These can easily be incorporated into the flow based algorithm, provided certain basic invariants about distributed edges is maintained (see below).

This completes the description of the backbone of the basic FA DR-planner. Next we consider some details ensuring the properties (a) – (c) above.

2.2.1 Ensuring dof completeness

First we intuitively explain why dof completeness is a crucial property. In Figure 7, after C_1 and C_2 are found, when C_1 is distributed, C_1 and C_2 would be picked up as a cluster, although they do not form a cluster. The problem is that the overconstrained subgraph W intersects C_1 on a trivial cluster, and W itself has not been found. Had W been found before C_1 was distributed, W would have been simplified into a wellconstrained subgraph and this misclassification would not have occurred.

It has been shown in [21] that this type of misclassification can be avoided (W can be forced to be found after C_2 is found), by maintaining three invariants. The first two are described here. The third is highly related to property (b) and is described in the next subsection.

The first is the following invariant: always distribute all undistributed edges connecting a new found cluster C (or the last distributed vertex that caused C to be found), to all the vertices distributed so far that are outside the cluster C. Undistributed weight on edges inside C are less crucial: if they become internal edges of the cluster, then this undistributed weight "disappears" when C is simplified into a wellconstrained cluster; there is also a simple method of treating undistributed weight on frontier edges so that they also do not cause problems - the method and proof can be found in [21]).

The second invariant that is useful for ensuring dof completeness is that for any cluster in the DR-plan, no

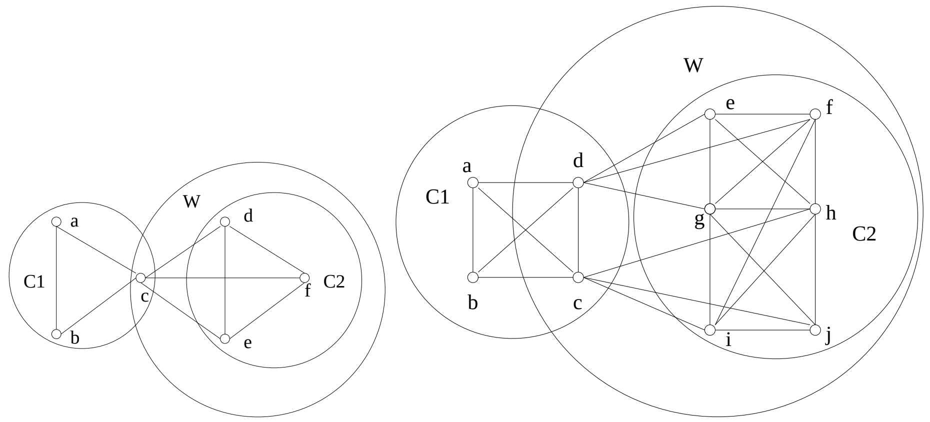


Figure 7: Finding W first will prevent dof misclassification: Left 2D example, Right 3D example.

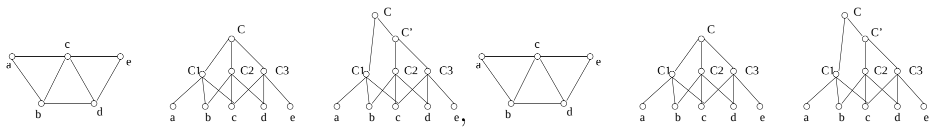


Figure 8: Ensuring Cluster Minimality: E is a set of essential clusters that must be present in any subset of the children of C that form a cluster. In this case, E itself forms a cluster. C' is a cluster made up of a proper subset of at least 2 of C's children

proper subset of atleast 2 of its child clusters forms a cluster. We call this property *cluster minimality.* FA ensures this using a generalization of the method *Minimal* of [13, 14] which finds a minimal dense subgraph inside a dense subgraph located by *DistributeVertex* and *DistributeEdge*.

See Figure 8. Once a cluster C is located and has children $C_1, \dots, C_k$, for $k \geq 2$, a recursive method *clusMin* removes one cluster C_i at a time (replacing earlier removals) from C and redoes the flow inside the flow graph restricted to C, before C's simplification. If a proper subset of atleast 2 C_j's forms a cluster C', then the clusMin algorithm is repeated inside C' and thereafter in C again, replacing the set of child clusters of C that are inside C' by a single child cluster C'. If instead no such cluster is found, then the removed cluster C_i the *essential*. I.e., it belongs to every subset of C's children that forms a cluster. When the set of clusters itself forms a cluster E (using a dof count), then *clusMin* is called on C again with a new child cluster E replacing all of C's children inside E.

2.2.2 (b) Finding a complete set of maximal clusters in underconstrained graphs

While the DR-planner described so far guarantees that at termination, top level clusters of the DR-plan are maximal. It also guarantees that the original graph is dof underconstrained only if there is more than one top level cluster in the DR-plan. However, in order to guarantee that *all* the maximal clusters of an underconstrained graph appear as top level clusters of the DR-plan, we use the observation that any pair of such clusters intersect on a subgraph that reduces (once incidence constraints are resolved) into a trivial subgraph (a single point in 2D or a single edge in 3D). This bounds the total number of such clusters and gives a simple method for finding all of them. Once the DR-planner terminates with a set of maximal clusters, other maximal clusters are found by simply performing a *Pushoutside* of 2 units on every vertex (in 2D) or every vertex and edge (in 3D), and continuing with the original DR-planning process until it terminates with a larger set of maximal clusters. This is performed for each vertex in 2D and each edge in 3D which guarantees that all maximal clusters will be found. See [21] for proofs.

2.2.3 (c) Controlling width of the DR-plan

FA achieves a linear bound on DR-plan width by maintaining the following invariant of the cluster or flow graph: *every pair of clusters in the flow graph (top level of the DR-plan) at any stage intersect on at most a trivial*

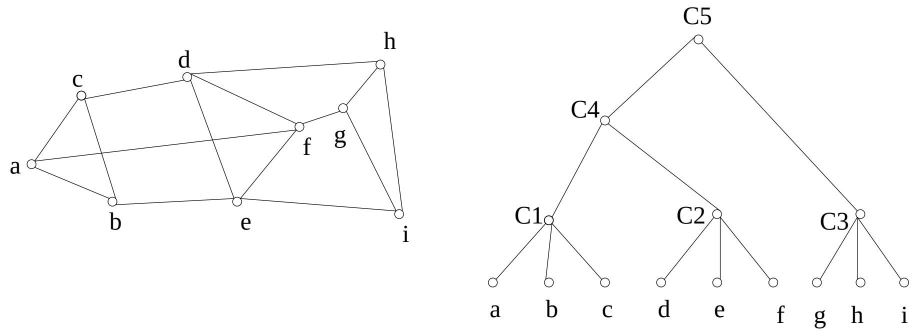


Figure 9: Prevent accumulation of clusters

subgraph. FA does this by repeatedly performing 2 operations each time a new potential cluster is isolated.

The first is an *enlargement* of the found cluster. In general, a new found cluster N is enlarged by any cluster D_1 currently in the flow graph, if their nonempty intersection is *not* a rotationally symmetric or trivial subgraph. In this case, N neither enters the cluster graph nor the DR-Plan. Only $N \cup D_1$ enters the DR-plan, as a parent of both D_1 and the other children of N. It is easy to see that the *sizes* of the subsystems corresponding to both $N \cup D_1$ and N are the same, since D_1 would already be solved.

For the example in Figure 9, when the DR-plan finds the cluster C_2 after C_1, the DR-planner will find that C_1 can be enlarged by C_2 The DR-planner forms a new cluster C_4 based on C_1 and C_2 and puts C_4 into the cluster queue, instead of putting C_2 to cluster queue.

The second operation is to iteratively *combine* $N \cup D_1$ with any clusters $D_2, D_3, \ldots$ based on a nonempty overlap that is not rotationally symmetric or trivial. In this case, $N \cup D_1 \cup D_2$, $N \cup D_1 \cup D_2 \cup D_3$ etc. enter the DR-plan as a staircase, or chain, but only the single cluster $N \cup D_1 \cup D_2 \cup D_3 \cup \ldots$ .. enters the cluster graph after removing D_1, D_2, D_3 $\ldots$.

Ofcourse, both of these processes are distinct from the original flow distribution process that *locates* clusters.

3 Module-Rigidity: Characterization and Algorithm

We give a recursive definition of 3D module-rigidity (along with a definition of module-complete DR-plans) and show that it is a natural and robust characterization. Then we sketch an extension of the FA algorithm in order to determine module-rigidity by constructing module-complete DR-plans. We follow with a number of examples of graphs that are dof rigid but not module-rigid.

Let C be a 3D distance constraint graph. Let $E, C_1, \ldots, C_k$ be proper subgraphs of C. We say that $C_1, \ldots, C_k \Rightarrow^{r,C} E$ (read: *implies rigidity of*) if by making $C_1, \ldots, C_m$ complete graphs (by adding additional edges), E becomes rigid. Analogously, we define $\Rightarrow^{d}$ and $\Rightarrow^{m}$ by asserting dof rigidity and module-rigidity (to be defined below) as the right hand side of the implication, respectively.

Let C be a 3D distance constraint graph. C is *module-rigid* if:
Base case: it is trivial and dof rigid.
Or the following holds. Let $Q = \{C_1, \ldots, C_m\}$ be any *complete, maximal, module decomposition* of C. This is defined as follows. Let $\phi^{m,C,*}$ be the transitive closure of the empty set under $\Rightarrow^{m,C}$, i.e., if there is a proper subgraph E of C s.t. either it is module-rigid, or there is some set of module-rigid proper subgraphs $C_1, \ldots, C_k$ of C s.t. $C_1, \ldots, C_k \Rightarrow^{m,C,*} E$, then E belongs to $\phi^{m,C,*}$. Let Q be any subset $\{C_1, \ldots, C_m\}$ of $\phi^{m,C,*}$ s.t.
Case 1: $m = 2$; C_1 and C_2 intersect on a nontrivial subgraph and their union induces all of C. Or, the following holds.
Case 2:

- Any maximal subgraph in $\phi^{m,C,*}$ must contain one of the C_i in Q.
- Each of the C_i's is *nearly maximal* with respect to the set Q in the following sense: the only elements of $\phi^{m,C,*}$ that strictly contain C_i intersect all the other subgraphs C_j, $j \neq i$ on nontrivial subgraphs.

Then C is module-rigid if

$$\sum_{S\subseteq Q}(-1)^{|S|-1}Adj - dof(\bigcap_{C_i\in S}(C_i)) \leq D,$$

where (recall) D is the number of dofs of a rigid body, and *Adj-dof*(C_i) is either the number of dofs (negation of density) of C_i if C_i is trivial; or simply D if C_i is nontrivial.

Observation 3.1 *Every module-rigid graph is dof rigid; and every rigid graph is module-rigid.*

The next lemma shows the tractability of the above characterization.

Lemma 3.2 *Let $Q = \{C_1, \ldots , C_m\}$ be any set of proper subgraphs of C that form a complete, maximal, module decomposition of C. This implies that if $m > 2$, then no pair of C_i intersect on more than a trivial subgraph. Thus m is at most $O(n^3)$, where n is the number of vertices C. Thus, the computation of the inclusion-exclusion formula in Definition 3 takes $O(n^3)$ time.*

Using the above lemma, the following definition shows the use of so-called module-complete DR-plans to efficiently determine module rigidity.

A *module* DR-plan P for a 3D distance constraint graph G is a partial order where each node represents a subgraph in $\phi^{m,G,*}$ or G itself, if G is module rigid. The ordering is by containment. These nodes are called *module clusters* (to be crucially differentiated from module-rigid subgraphs of G, which we call *inherent module clusters*). The leaves are the original vertices of G. Each node in the subDR-plan rooted at a node C represents a subgraph in $\phi^{m,C,*}$ or C itself, if C is module-rigid, or an inherent module cluster. If G is module-rigid, there is a single source cluster; if G is not module-rigid, the roots or sources form is a complete, maximal, module decomposition of G. A partial module DR-plan does not have to satisfy the conditions on the roots or sources. A module DR-Plan is *module-complete* if the set Q of child clusters of every module cluster C in P is a complete, maximal, module decomposition of C.

A module DR-plan is typically defined to contain additional information by incorporating another partial order called the *solving priority order*, which is consistent with the module DR-plan's DAG order, but could be more refined. The intent is that module-rigidity of module clusters that appear later in the order depend on clusters that appear earlier. I.e., the ordering reflects the number of applications of $\Rightarrow^m$ required to find a module cluster.

The next theorem shows that module-rigidity is robust. I.e., the order of bottom-up construction of module(-complete) DR-plans is immaterial, a type of Church-Rosser property.

Theorem 3.3 *If a graph G is module-rigid, then every partial, module(-complete) DR-plan for G can be extended to a module(-complete) DR-plan.*

Modification of FA to determine module rigidity

The above discussion effectively lays out a tractable method for determining module-rigidity by computing module-complete DR-plans bottom up. This is done by extending the dof-complete DR-planner FA given in the previous section as follows. First note that by using FA we guarantee no false negatives, since module-rigid implies dof-rigid. We now sketch how to eliminating dof-rigid graphs that are not module-rigid. The FA DR-planner running on an input graph G uses DistributeCluster (flow) on the current set S of dof rigid clusters to isolate a dof cluster candidate C, and thereafter constructs a a complete, maximal, dof rigid decomposition of it, after which it decides whether a new dof-rigid cluster C has been found, using the dof-rigid characterization of Section 2. This is the key point of extension. We use the analogy between this dof-rigid characterization and the module-rigid characterization at the beginning of this Section. Inductively, we can assume that the current set S consists of module-rigid subgraphs of G or inherent module clusters. The method of construction of a complete, maximal, module decomposition Q of the candidate (inherent module) cluster C can be done without constructing $\phi^{m,C,*}$, by constructing a sequence of Q^i's each of which satisfies the above conditions on Q, but with respect to $\phi^{m,C,i}$ (i.e, closure with respect to i applications of the $\Rightarrow^m$ operation). This sequence reaches a fixed point at Q.

We give examples that illustrate the use of module-rigidity.

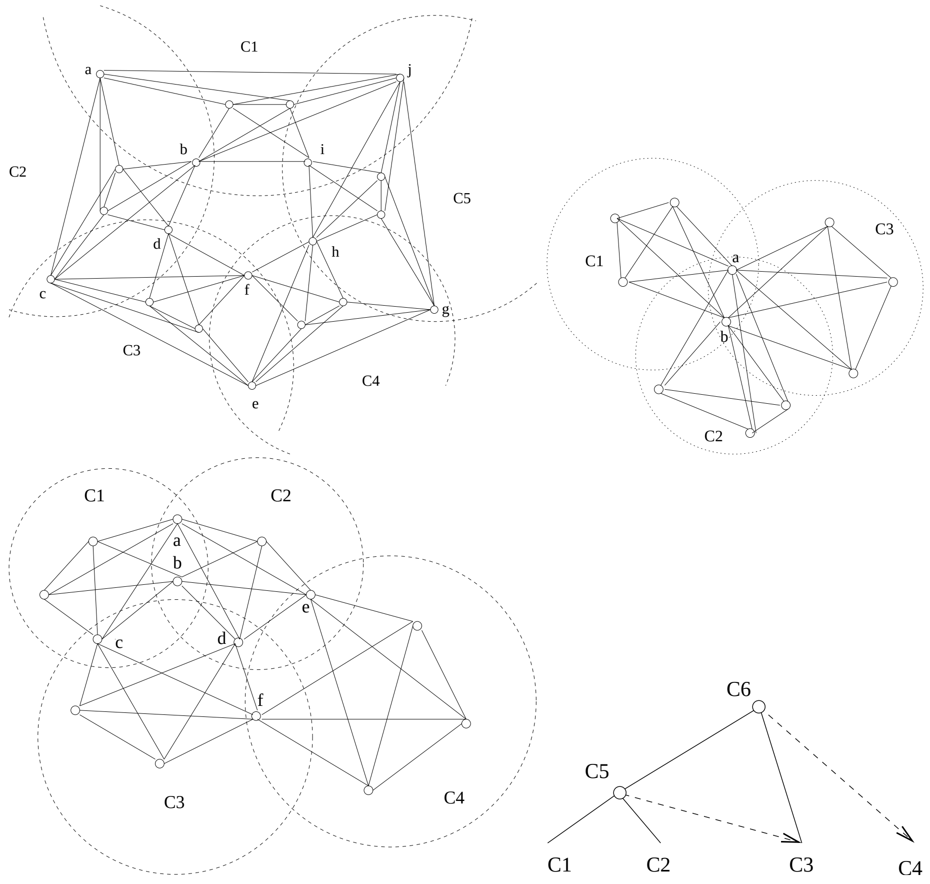


Figure 10: Examples where module-rigidity beats dof rigidity. See text for explanation.

In Figure 10 Top Left: the graph is dof rigid, but not module-rigid, as seen by the complete maximal module decomposition shown. Top Right: module-rigid, but no pair of inherent module clusters shown forms a module-rigid subgraph, they do form dof-rigid subgraphs. Bottom: not module-rigid but dof rigid; complete maximal module decomposition and solving priority orders as follows: the pair C_1, C_2 is an inherent module cluster C_5 but that C_5 can be solved only after C_3 is solved; I.e., before the virtual edge (c, d) is added, C_1 and C_2 would not be picked up together as a cluster candidate. Similarly, it will also determine that C_5, C_3 form a cluster C_6, but solving priority order shown. Bottom Right: module-complete DR-plan for left constraint system with 2 sources or roots: C_6 and C_4.

Figure 11 shows a classic graph from [4, 5], with “hinges,” which is not module-rigid but is dof rigid. A complete maximal, module decomposition is shown. The middle cluster C_2 is not an inherent module cluster, although C_1 and C_2 are.

Open Problem

A question that immediately arises is to relate the characterization given here to rigidity matroids and standard conjectures on combinatorial rigidity characterizations for 3D [8].

References

[1] S. Ait-Aoudia, R. Jegou, and D. Michelucci. Reduction of constraint systems. In *Compugraphics*, pages 83–92, 1993.

[2] B. Bruderlin. Constructing three-dimensional geometric object defined by constraints. In *ACM SIGGRAPH*. Chapel Hill, 1986.

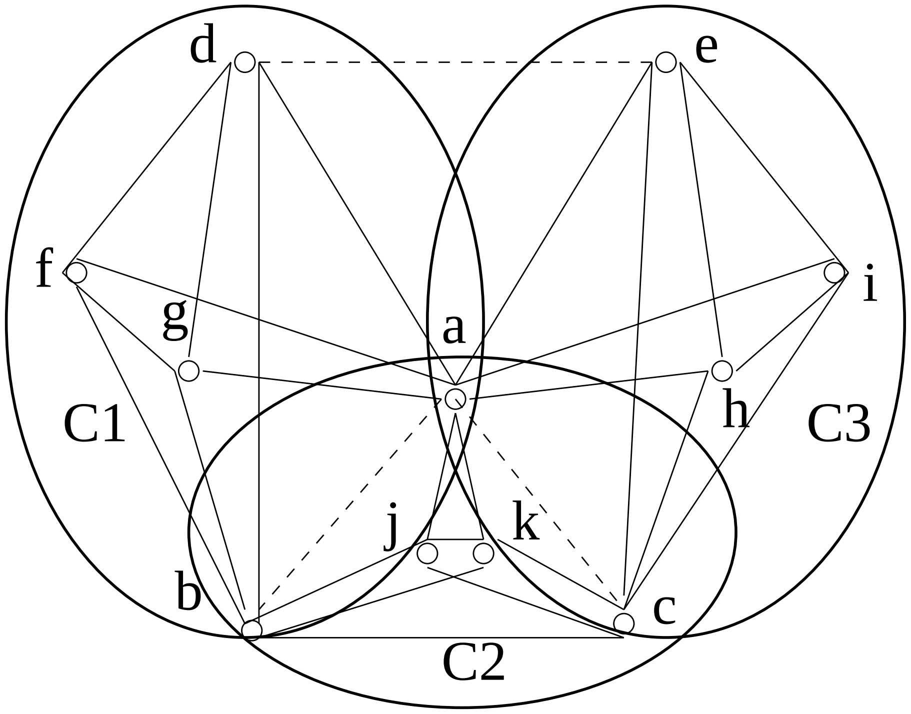


Figure 11: Classic Hinge example: not module-rigid, but dof rigid

.

[3] D. Cox, J. Little, and D. O'Shea. *Using algebraic geometry.* Springer-Verlag, 1998.

[4] Henry Crapo. Structural rigidity. *Structural Topology*, 1:26–45, 1979.

[5] Henry Crapo. The tetrahedral-octahedral truss. *Structural Topology*, 7:52–61, 1982.

[6] I. Fudos. *Geometric Constraint Solving.* PhD thesis, Purdue University, Dept of Computer Science, 1995.

[7] I. Fudos and C. M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, 16:179–216, 1997.

[8] Jack E. Graver, Brigitte Servatius, and Herman Servatius. *Combinatorial Rigidity.* Graduate Studies in Math., AMS, 1993.

[9] B. Hendrickson. Conditions for unique graph realizations. *SIAM J. Comput.*, 21:65–84, 1992.

[10] C Hoffman, M Sitharam, and B Yuan. Making constraint solvers more useable: the overconstraint problem. *to appear in CAD*, 2004.

[11] C. M. Hoffmann, A. Lomonosov, and M. Sitharam. Finding solvable subsets of constraint graphs. In Smolka G., editor, *Springer LNCS 1330*, pages 463–477, 1997.

[12] C. M. Hoffmann, A. Lomonosov, and M. Sitharam. Geometric constraint decomposition. In Bruderlin B. and Roller D., editors, *Geometric Constr Solving and Appl*, pages 170–195, 1998.

[13] Christoph M. Hoffmann, Andrew Lomonosov, and Meera Sitharam. Finding solvable subsets of constraint graphs. In *Constraint Programming '97 Lecture Notes in Computer Science 1330, G. Smolka Ed., Springer Verlag*, Linz, Austria, 1997.

[14] Christoph M. Hoffmann, Andrew Lomonosov, and Meera Sitharam. Geometric constraint decomposition. In Bruderlin and Roller Ed.s, editors, *Geometric Constraint Solving.* Springer-Verlag, 1998.

[15] Christoph M. Hoffmann, Andrew Lomonosov, and Meera Sitharam. Planning geometric constraint decompositions via graph transformations. In *AGTIVE '99 (Graph Transformations with Industrial Relevance), Springer lecture notes, LNCS 1779, eds Nagl, Schurr, Munch*, pages 309–324, 1999.

[16] Christoph M. Hoffmann, Andrew Lomonosov, and Meera Sitharam. Decomposition of geometric constraints systems, part i: performance measures. *Journal of Symbolic Computation*, 31(4), 2001.

[17] Christoph M. Hoffmann, Andrew Lomonosov, and Meera Sitharam. Decomposition of geometric constraints systems, part ii: new algorithms. *Journal of Symbolic Computation*, 31(4), 2001.

[18] G. Kramer. *Solving Geometric Constraint Systems.* MIT Press, 1992.

[19] G. Laman. On graphs and rigidity of plane skeletal structures. *J. Engrg. Math.*, 4:331–340, 1970.

[20] R. Latham and A. Middleditch. Connectivity analysis: a tool for processing geometric constraints. *Computer Aided Design*, 28:917–928, 1996.

[21] Andrew Lomonosov and Meera Sitharam. Graph algorithms for geometric constraint solving. In *submitted*, 2004.

[22] A. Middleditch and C. Reade. A kernel for geometric features. In *ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications.* ACM press, 1997.

[23] J. J. Oung, M. Sitharam, B. Moro, and A. Arbree. Frontier: fully enabling geometric constraints for feature based design and assembly. In *abstract in Proceedings of the ACM Solid Modeling conference*, 2001.

[24] J. Owen. www.d-cubed.co.uk/. In *D-cubed commercial geometric constraint solving software.*

[25] J. Owen. Algebraic solution for geometry from dimensional constraints. In *ACM Symp. Found. of Solid Modeling*, pages 397–407, Austin, Tex, 1991.

[26] J. Owen. Constraints on simple geometry in two and three dimensions. In *Third SIAM Conference on Geometric Design.* SIAM, November 1993. To appear in Int J of Computational Geometry and Applications.

[27] J.A. Pabon. Modeling method for sorting dependencies among geometric entities. In *US States Patent 5,251,290*, Oct 1993.

[28] M Sitharam. Frontier, an opensource 3d geometric constraint solver: algorithms and architecture. *monograph, in preparation*, 2004.

[29] M Sitharam. Graph based geometric constraint solving: problems, progress and directions. In Dutta, Janardhan, and Smid, editors, *AMS-DIMACS volume on Computer Aided Design*, 2004.

[30] M Sitharam and M Agbandje-Mckenna. A geometry and tensegrity based virus assembly pathway model. *submitted, available upon request*, 2004.

[31] M Sitharam, A Arbree, Y Zhou, and N Kohareswaran. Solution management and navigation for 3d geometric constraint systems. *submitted, available upon request*, 2004.

[32] M Sitharam, J Oung, and A Arbree. Efficient underconstrained completions, updates and on line solution of general geometric constraint graphs. *submitted, available upon request*, 2004.

[33] M Sitharam, J Peters, and Y Zhou. Solving minimal, wellconstrained, 3d geometric constraint systems: combinatorial optimization of algebraic complexity. *submitted to ADG 2004, available upon request*, 2004.

[34] M Sitharam and Y Zhou. Mixing features and variational constraints in 3d. *submitted, available upon request*, 2004.

[35] Meera Sitharam. Frontier, opensource gnu geometric constraint solver: Version 1 (2001) for general 2d systems; version 2 (2002) for 2d and some 3d systems; version 3 (2003) for general 2d and 3d systems. In *http://www.cise.ufl.edu/∼sitharam, http://www.gnu.org*, 2004.