Geometric constraints within Feature Hierarchies

Meera Sitharam*† Jian-Jun Oung* Yong Zhou* Adam Arbree*

April 18, 2005

Abstract

We study the problem of enabling general 2D and 3D variational constraint representation to be used in conjunction with a feature hierarchy representation, where some of the features may use procedural or other non-constraint based representations. We trace the challenge to a requirement on constraint decomposition algorithms or decomposition-recombination (DR) planners used by most variational constraint solvers, formalize the feature hierarchy incorporation problem for DR-planners, clarify its relationship to other problems, and provide an efficient algorithmic solution. The new algorithms have been implemented in the general, 2D and 3D opensource geometric constraint solver FRONTIER developed at the University of Florida.

Keywords: Variational geometric constraint solving, Feature-based and assembly modeling, Conceptual design, Cyclical and 3D geometric constraint systems, Decomposition of geometric constraint systems, Underconstrained and Overconstrained systems, User navigation of solution conformations, Parametric constraint solving, Degree of Freedom analysis, Constraint graphs.

1 Introduction and Motivation

Designers find it intuitive to use a spatial feature hierarchy representation, which includes: a procedural history or an almost linear sequence of attachments, extrusions, sweeps; or CSG Boolean operations such as intersections; or parametric constraints, while additionally permitting B-rep and other representations of some features. (We use the FEMEX and other standard definitions of feature hierarchy, [5], [2] and are concerned primarily with the conceptual design stage).

While designers additionally appreciate the expressiveness of variational constraints, today's CAD systems largely restrict variational constraint representations to 2D cross sections. This persists despite the general consensus that advocates a judicious use of 3D variational constraints for the intuitive expression and maintanence of certain complex and cyclic relationships that often occur between features, parts or subassemblies.

To rectify this situation, it would be desirable if a feature hierarchy could simultaneously incorporate 2D and 3D variational constraints. In this paper, we denote such representations as *mixed* representations. See Figure 2.

Previous work on such mixed representations can be classified into two broad types. The first type, such as [26, 27, 9], dictates a unified representation language which is an amalgamation of variational constraints with other representation languages such as CSG and Brep. The second type, such as [14] wrestles with a heterogeneous approach, using many servers, one for each representation language, so that the appropriate one can be called when required. Both approaches, while highly general in scope, have their drawbacks.

Our approach in this paper has a narrower focus: how to freely enable a variational constraint solver to deal with feature interactions at any level of a feature hierarchy, permitting the features to be independently manipulated. Recursively, these features could themselves be represented using other representations, or in a similar, mixed manner, using constraints to relate the sub-features. This is a natural representation, since regardless of the way in which the features at any given level are represented,

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

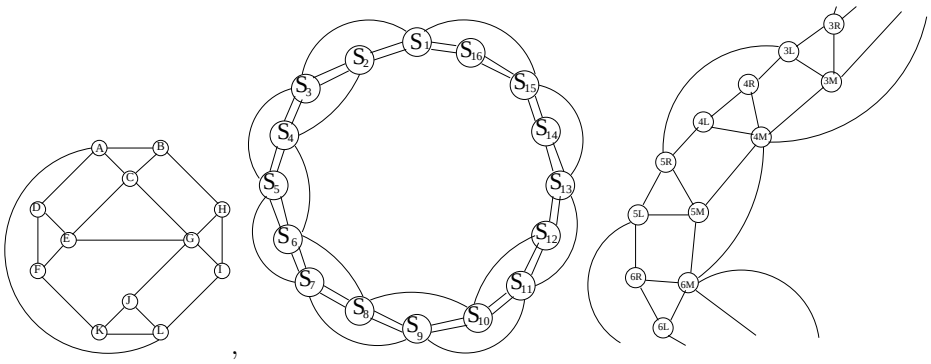

Figure 1: Left: Small 2D constraint graph (vertices are points with 2 dofs and edges are distance constraints removing 1 dof) that is not triangle decomposable, in fact, has no DR-plan of size 3. Middle: Arbitrarily large non-triangle-decomposable constraint graph; Each S_i consists of 2 adjacent triangles (also of vertices representing points and edges representing distance constraints) in the magnification on right.

constraints between features at that level could be specified between primitive geometric objects or *handles* belonging to the features. See Figure 2.

One main challenge to achieving this type of representation is the following competing pair of requirements on variational constraint solvers.

• Effectively representing the interaction of independent features, especially in 3D, would require the use of fairly general, cyclic constraint systems, which are often (consistently) overconstrained.

To deal with and resolve such constraint systems the efficient generation of a close-to-optimal *decomposition and recombination (DR) plan* (formally defined in Section 2 of *general* 2D or 3D constraint graphs is needed. These may not be amenable to decomposition into triangular or other fixed patterns which many DR-planners use. See for example Figure 1.

DR-plans are widely used in geometric constraint solving and are crucial in order to deal with the tractability bottleneck in constraint solving: minimizing the size of simultaneous polynomial equation systems, thereby controlling the dependence on exponential time algebraic-numeric solvers which are practically crippled when dealing with even moderately sized systems. Effective DR-plans are used also for navigating the solution space of the constraint system [40], optimizing the algebraic complexity of subsystems sent to an algebraic-numeric solver [41], for dealing with explicit inconsistencies (overconstrained systems) [15], implicit or geometric redundancies and constraint dependences [42], [43], ambiguities (underconstrained systems) and for efficient updates and online solving [38, 39].

• In order for variational constraints to be used in conjunction with a feature hierarchy representation, the DR-plans of geometric constraint systems should now be made to incorporate an *input, feature decomposition* representing the underlying feature, part or subassembly hierarchy. This is a partial order, typically represented as a Directed Acyclic Graph or *dag*. See Figures 2, 3, 4.

Again, this incorporation of an arbitrary input, conceptual design decomposition into the DR-plan is only possible if the DR-planning process is sufficiently general. In particular, the incorporation should be insensitive to the order in which the elements in the constraint graph are considered during the DR-planning process, the so-called Church-Rosser property. However this property while necessary, is not sufficient as explained in Section 2). The paper [22] first formalized the concept of a DR-plan as well as several performance measures of DR-planners some of which are relevant to this paper. It additionally gives a table of comparisons which shows that many of the previous DR-planners e.g.[3, 35, 36, 3], [24, 25, 11, 12], [1, 37, 30, 28], (or any obvious modifications of them) would inherently fail to incorporate even tree-like input design decompositions or feature hierarchies. We explain in Section 3 what the difficulty is in the case of DR-planners that depend on decomposition based on fixed patterns.

Incorporation of input decompositions is crucial also in order to capture design intent, or assembly order and allow independent and local manipulation of features, parts, subassemblies or subsystems within their local coordinate systems. It is also crucial for facilitating solution space navigation [40] in a

manner that reflects a conceptual design intent. In addition, it is crucial for providing the user with a feature repertoire, the ability to paste into a sketch already resolved features and constraint subsystems that are specified in another representation or allowing certain features or easier subsystems to be solved using other, simpler, methods such as triangle-based decomposition or parametric constraint solving. Sometimes the user would prefer to specify a priority at the vertices of the dag which dictates the order of resolution of the features, parts, subassemblies or subsystems. This occurs also when one features can only be defined or generated based on another, as in procedural or history based representations. Also, note that parametric constraint solving can in fact be achieved as a special case, where the order is a complete, total order. More generally, this can be used in the CAD database maintanence of multiple product views as in [16], [8], for example the design view and a downstream application client's view may be somewhat different constraint systems and the two feature hierarchies may not even be refinements of one another, but intertwined. See Figure 4. Furthermore, each view could contain different referencing shape elements that are not part of the net shape and therefore, not part of the other views. This is particularly the case when these referencing shape elements are actually generated during the operations of a history-based procedural representation.

1.1 Contribution and Organization

Our starting point is the recently developed *Frontier Vertex Algorithm (FA)*, a DR-planner [19] [20], [22], [32], [15], [42], which builds upon nearly a decade of earlier work on geometric constraint solving and deals with general 2D and 3D variational constraint systems.

We give a feature incorporation algorithm that sits atop the FA DR-planner and permits it to incorporate input design decompositions and feature hierarchies, while preserving its efficiency and other desirable properties. The new algorithm has been implemented in the opensource 2D and 3D geometric constraint solver FRONTIER [45], [33], [39].

In Section 2, we give the necessary background on geometric constraint graphs, degree of freedom analysis, its limitations, DR-plans and their essential and desirable properties. Section 3 gives a formal statement of the problem and differentiates it from another, similar problem. Section 4 gives those algorithmic essentials of the FA DR-planner that are absolutely necessary to give our feature incorporation algorithm which is described in Section 5. A pseudocode is provided in [39] and the documented code can be found in [45].

2 Basic Background

Geometric constraint systems have been studied in the context of variational constraint solving in CAD for nearly 2 decades For recent reviews of the extensive literature on geometric constraint solving more elaborate descriptions and examples for the definitions below, see, e.g, [20, 28, 10, 38].

A *geometric constraint system* consists of a finite set of primitive geometric objects such as points, lines, planes, conics etc. and a finite set of geometric constraints between them such as distance, angle, incidence etc. The constraints can usually be written as algebraic equations and inequalities whose variables are the coordinates of the participating geometric objects. For example, a distance constraint of d between two points (x_1, y_1) and (x_2, y_2) in 2D is written as $(x_2-x_1)^2+(y_2-y_1)^2 = d^2$. In this case the distance d is the *parameter* associated with the constraint. Most of the constraint solvers so far deal with 2D constraint systems. With the exception of work [17, 18, 22, 23], [21], [15, 31, 32, 33, 39, 42, 40, 41], related to the FRONTIER geometric constraint solver [45], to the best of our knowledge, work on standalone 3D geometric constraint solvers is relatively sparse [4, 34].

A *solution or realization* of a geometric constraint system is the (set of) real zero(es) of the corresponding algebraic system. In other words, the solution is a class of valid instantiations of (the position, orientation and any other parameters of) the geometric elements such that all constraints are satisfied. Here, it is understood that such a solution is in a particular geometry, for example the Euclidean plane, the sphere, or Euclidean 3 dimensional space. A constraint system can be classified as *overconstrained*, *wellconstrained*, or *underconstrained*. Well-constrained systems have a finite, albeit potentially very large number of *rigid* solutions; i.e., solutions that cannot be infinitesimally flexed to give another nearby

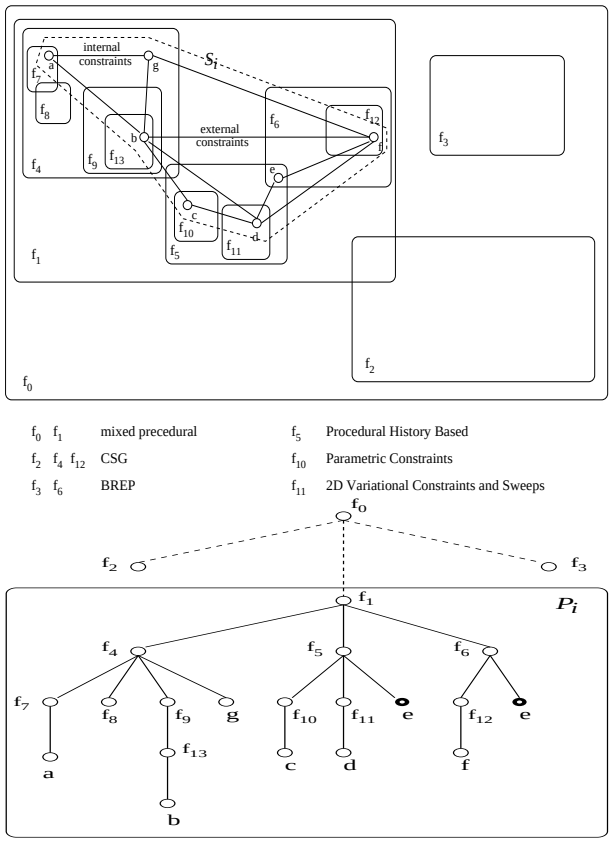

Figure 2: Constraint system S_i and underlying feature hierarchy P_i. Features $f_0, \ldots, f_{11}$ are in mixed representation; a, b, c, d, e are participating feature handles; $f_0, \ldots, f_3$ are features at a higher level of hierarchy

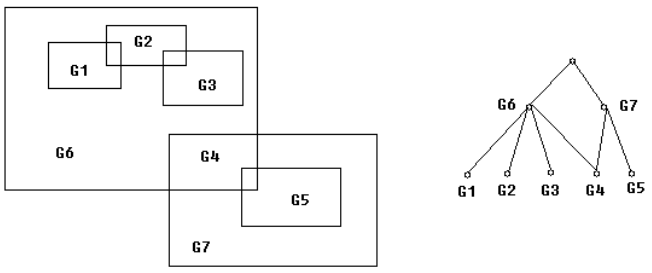

Figure 3: Input design decomposition or feature hierarchy and corresponding directed acyclic graph on right.

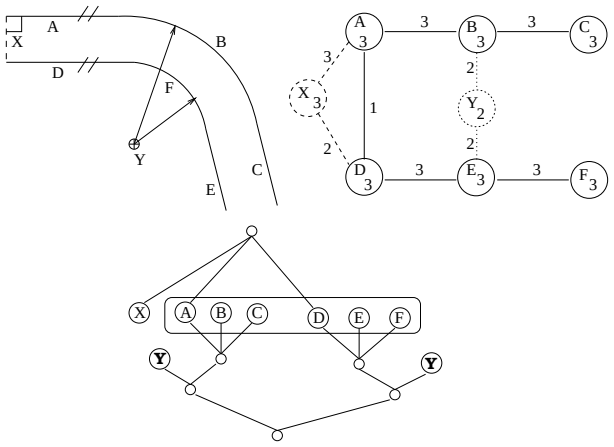

Figure 4: Above: Constraint system and constraint graph showing multiple views (dotted and dashed) with different referencing elements x and y; numbers represent dofs. Below: the two views have intertwined feature hierarchies; box contains net shape elements common to both views

solution: the solution space (modulo rigid body transformations such as rotations and translations) consists of isolated points - it is zero-dimensional. Underconstrained systems have infinitely many solutions; their solution space is not zero-dimensional. Overconstrained systems do not have a solution unless they are *consistently overconstrained*. In that case, they could be embedded within overall underconstrained systems. Systems that are not underconstrained are called *rigid* systems.

2.1 Constraint Graphs and Degrees of Freedom

DR-plans, formally defined in the next section, provide the formal basis of our feature hierarchy incorporation algorithm. Geometric constraint graph representations of a constraint system are typically used to develop DR-plans. Specifically these graphs are used for combinatorial analysis of algebraic properties of the system (such as wellconstrainedness, rigidity etc.,), that hold *generically*, i.e., for all generic values for the constraint parameters (for example, for almost all distance values, in the case of distance constraint systems).

Please see, for example, [20] for more details related to the definitions given below. A geometric constraint graph $G = (V, E, w)$ corresponding to geometric constraint system is a weighted graph with vertex set (representing geometric objects) V and edge set (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.

For example, Figures 5, 6, 9 show 2D and 3D constraint systems and their respective dof constraint graphs. More 3D constraint systems whose graphs have vertices of weight 3 (points) and edges of weight 1,3 can be found in Figures 7 8.

Note that the constraint graph could be a *hypergraph*, each hyperedge involving any number of vertices. 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.

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 spatial contexts $D = 6$, in general. If we expect the rigid body to be fixed with respect to a global coordinate system, then $D = 0$. A *trivial* subgraph is single vertex (in 2D) and a vertex or edge (in 3D).

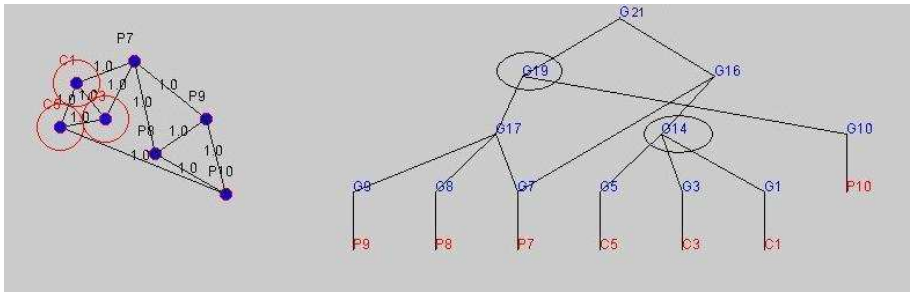

Figure 5: 2D constraint system example with variable radius circles, points and distance constraints - all equal distances. The corresponding graph has 7 vertices, 3 vertices representing variable radius circles (each with 2 degrees of freedom for the center position) and 4 vertices representing points (each with 2 positional degrees of freedom). The edges are the distance constraints shown in the constraint system. Each edge has weight 1 since a distance constraint removes 1 degree of freedom from one of the two participating points.

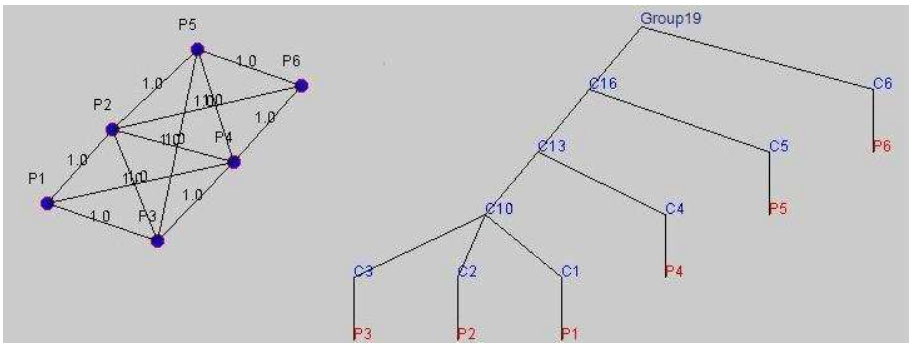

Figure 6: Simple 3D constraint system drawn on 2D canvas with points and distances and DR-plan

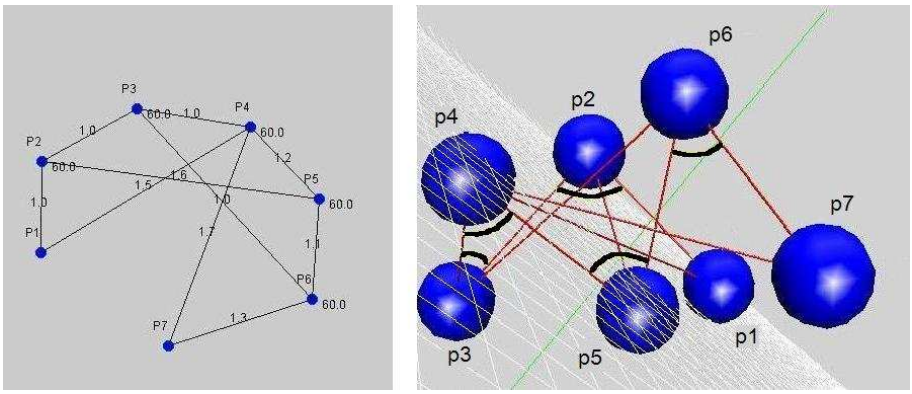

Figure 7: 3D constraint system drawn on 2D canvas with 5 point objects ($P2$ - $P6$) and 6 fixed length line segment objects (between point P_i and P_{i+1}); 4 distance constraints (equal distances); 10 incidence constraints where 2 linesegments are incident at each of the points $P2$ -$P6$; and 5 equal angle constraints between adjacent linesegments. The corresponding graph has 13 vertices of weight 3 (points and fixed length linesegments have 3 degrees of freedom each); edges of weight 3 (an incidence constraint between 2 points remove the 3 degrees of freedom of one of the points) and edges of weight 1 (a distance constraint between 2 points remove 1 degree of freedom from one of the points; an angle constraint between 2 line segments removes 1 degree of freedom from one of the lines). Solution (right)

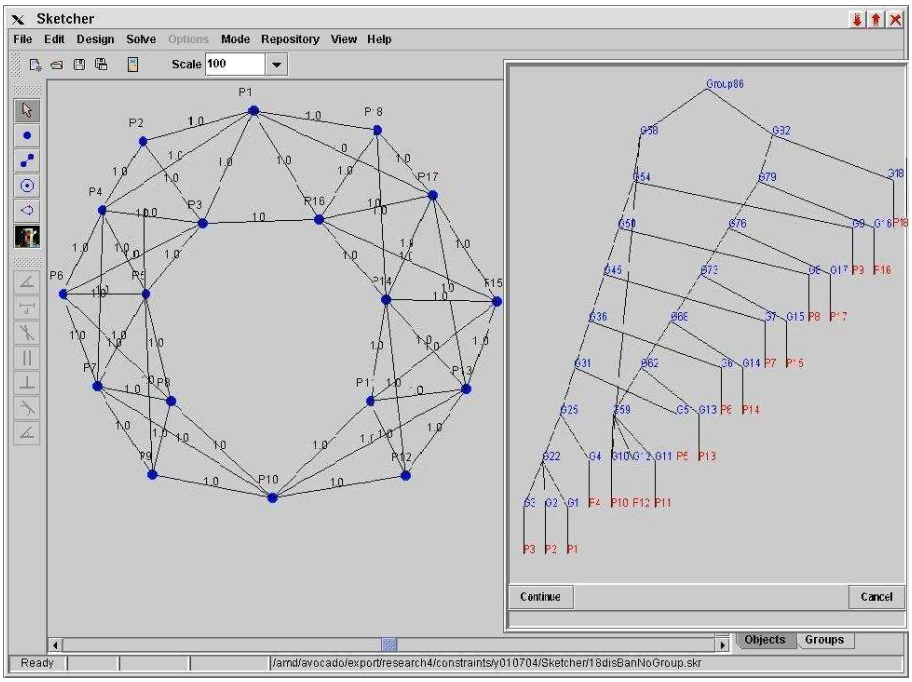

Figure 8: 3D constraint system with "bananas" type [13] generic constraint dependence not detectable by a simple dof count: the distance between P1 and P10 is independently determined by the left and right half-moon clusters. It is however well-overconstrained and consistently constrained for the given set of distances: corresponding DR-plan has single root

Next we give purely combinatorial properties related to density that are used to detect generic algebraic properties. A dense, nontrivial graph with density strictly greater than $-D$ is called *dof-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 overconstrained subgraph, and has the property that on replacing all overconstrained subgraphs by dof-wellconstrained subgraphs (in any manner), G remains dense. Intuitively, this definition is used to prevent some overconstrained subgraphs with high density from skewing the classification of the entire graph. In particular, an extreme example could be a graph that has 2 subgraphs that are severely overconstrained, but with no constraints between them. By this definition, such a graph would not be well-overconstrained and would be correctly classified as underconstrained. A graph that is wellconstrained or well-overconstrained is called *dof-cluster*. A nontrivial dense graph is *minimal* if it has no nontrvial dense proper subgraph. 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 *underconstrained*. If a dense graph is not minimal, it could in fact be an underconstrained graph: as pointed out, the density of the graph could be the result of embedding a subgraph of density greater than $-D$.

Next we discuss how the graph theoretic properties *degree of freedom (dof) analysis* relate to corresponding properties of the corresponding constraint *system*.

In 2 dimensions, according to Laman's theorem [29], if all geometric objects are points and all constraints are distance constraints between these points then any minimal dense cluster represents a generically rigid system. In general, however, while generically rigid system always gives a cluster, the converse is not always the case. In fact, there are wellconstrained, dense clusters whose corresponding systems are not generically rigid and are in fact generically not rigid due to the presence of generic *constraint dependences*. See Figure 8 with 3D points and distance constraints, which illustrates the so-called "bananas" problem of [13], which generalizes to the so-called "hinge" problem [6, 7]. To date, there is no known, tractable, combinatorial characterization of generic rigidity of systems for 3 or higher dimensions even when only points and distances are involved [46], [13], although several conjectures exist. There are no

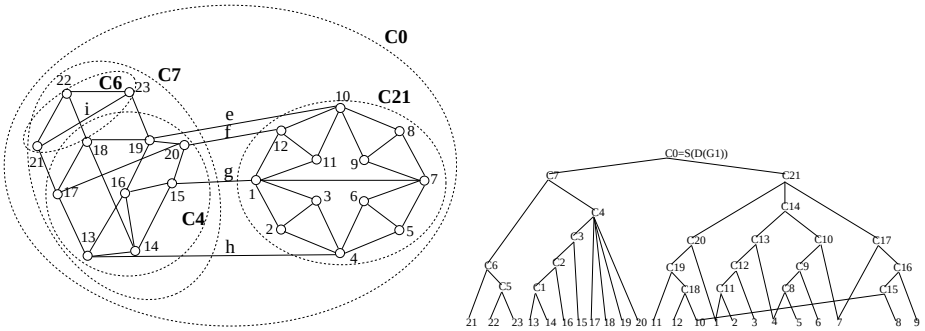

Figure 9: 2D constraint graph G1 and DR-plan; all vertices have weight 2 and edges weight 1

known general combinatorial characterizations of 2D rigidity, when other constraints besides distances (such as angles) are involved. For constraint systems with angle and incidence constraints, but no distances such a characterization is given in [43]. For 3D points and distances, the notion of *module-rigid clusters* in [42] (an extension of dof-rigid clusters defined above) deals with all aspects of the bananas and hinge problems, i.e., it correctly characterizes generic rigidity in all known cases. Currently, no counterexamples are known - of module-rigid constraint graphs that are not generically rigid.

Note. In this paper, we restrict our attention to dof-rigid clusters, and DR-planners and feature incorporation algorithms that are based on them. See discussion at the end of Section 5.

2.2 Formal definition of DR-plans and Essential Properties

Here we formally define and give essential properties of DR-plans and DR-planners. This section is important since we require our feature incorporation algorithm to preserve these properties. Please refer to [22, 31] for a detailed discussion of these properties.

Formally, a *dof-DR-plan* of a constraint graph G is a directed acyclic graph (dag) whose nodes represent dof-clusters in G, and edges represent containment. The leaves or sinks of the dag are all the vertices (primitive dof-clusters) of G. The roots or sources are a complete set of the maximal dof-clusters of G. For well or well-overconstrained graphs, the DR-plans have a single source. There could be many DR-plans for G. See Figures 9, 5, 6, 7, 8.

2.2.1 Overconstraints

First, each dof-cluster C in the DR-plan should be accompanied by a tractable representation of a complete list of *reducible* overconstraint sets directly associated with C. I.e., sets of constraints that do not lie entirely within any child cluster of C and can be removed without affecting the dof-rigidity of C. The DR-planner should additionally admit an efficient method of removing overconstraints by making the appropriate changes to the DR-plan.

2.2.2 Optimality

The *size* of a cluster in a DR-plan is its fan-in or number of its children (it represents the size of the corresponding subsystem, once its children are solved). Since the algebraic-numeric solvers take time exponential in the size of the subsystems they solve, and the number of solutions is also typically exponential, minimizing the size of a DR-plan is *essential* to the ESM method presented here. An *optimal* DR-plan is one that minimizes the maximum fan-in. It is shown in [31], [32], that the problem of finding the optimal DR-plan of even a 2D distance constraint gr aph is NP-hard, and approximability results are shown only in special cases. Nonapproximability results are not known.

One measure used in lieu of absolute optimality is based on the fact that 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.

A more satisfactory measure of optimality is based on the following alternative property. A *tractable* DR-plan for systematic navigation should ensure that each cluster C should be accompanied by a a small set of its children C_i that form an *optimal covering set* of maximal clusters properly contained in C. A *covering set* of clusters is one whose union contains all geometric elements within C. The size of C is simply the size of this optimal covering set. The *optimality* here refers not to the size of the covering set, but to any suitable combinatorial measure of the algebraic complexity of the active constraint system for solving C, given the solutions of the child clusters in the covering set. This leads to the notion of *completeness* of DR-plans, given below.

2.2.3 Completeness

Any method that chooses an optimal covering set for a cluster C requires as input a generalized *complete decomposition of C into maximal proper subclusters,* formally defined as follows. The decomposition of any cluster C falls into one of 2 types. A Type 1 cluster C has exactly 2 child clusters, which intersect on a nontrivial subgraph, and their union covers all the geometric elements in C. A Type 2 cluster C has a set of child clusters C_i with the following property. The union of C_i's covers all the geometric elements in C; any pair of C_i's intersect on at most a trivial subgraph; and every C_i is a *proper maximal* subcluster of C, i.e., there is no proper subcluster of C that strictly contains C_i. Completeness is also needed for detecting implicit constraint dependences and for more accurate, module-rigid DR-planners, see discussion at the end of Section 5.

2.2.4 Complexity

Another basic property of a DR-plan is its *width* i.e, *number* of clusters in the DR-plan to be small, preferably linear in the size of G: this reflects the complexity of the planning process and affects the complexity of the solving process that is based on the DR-plan. Clearly, this property competes with completeness.

Other desirable properties of DR-planners not mentioned above include systematic correction of under-constrained systems, and amenability to efficient updates of geometric primitives or constraints.

3 Formal Problem Statement

We now describe the algorithmic problem of *feature incorporation* in a DR-plan as an input-output specification.

Input:

(1) A 2D or 3D geometric constraint system (see Section 2), S.
(2) An *partial feature decomposition* of the constraint system given as a directed acyclic graph (dag) P obtained from one or more feature, part or subassembly hierarchies or design views. A feature or subassembly is a node of P with its immediate subfeatures represented as its children. See Figures 10, 3, 11.

Desired Output: A DR-plan for the input constraint system whose nodes include those features in the input partial feature decomposition that are dof-clusters. For features that are not dof-clusters, a complete set of maximal dof-clusters within them should appear as nodes in the DR-plan.

Other output Requirements: Once the output DR-plan has been obtained, updates to the input feature hierarchy should be achieved efficiently without having to redo the entire DR-plan.

The desirable properties of DR-plans and DR-planners given in Section 2 DR-planners) must be preserved by the feature incorporation algorithm and its output DR-plan. Some of these properties require straightforward re-definition for feature incorporating DR-plans, since often no DR-plan even exists that incorporates a given input feature decomposition and preserves these properties as such. For example, it may not be possible to preserve optimality since certain features may simply require clusters with large fan-in; similarly it may not be possible to preserve complexity and small width, since the number of given features may force a large width. However, we would like to ensure that our algorithm

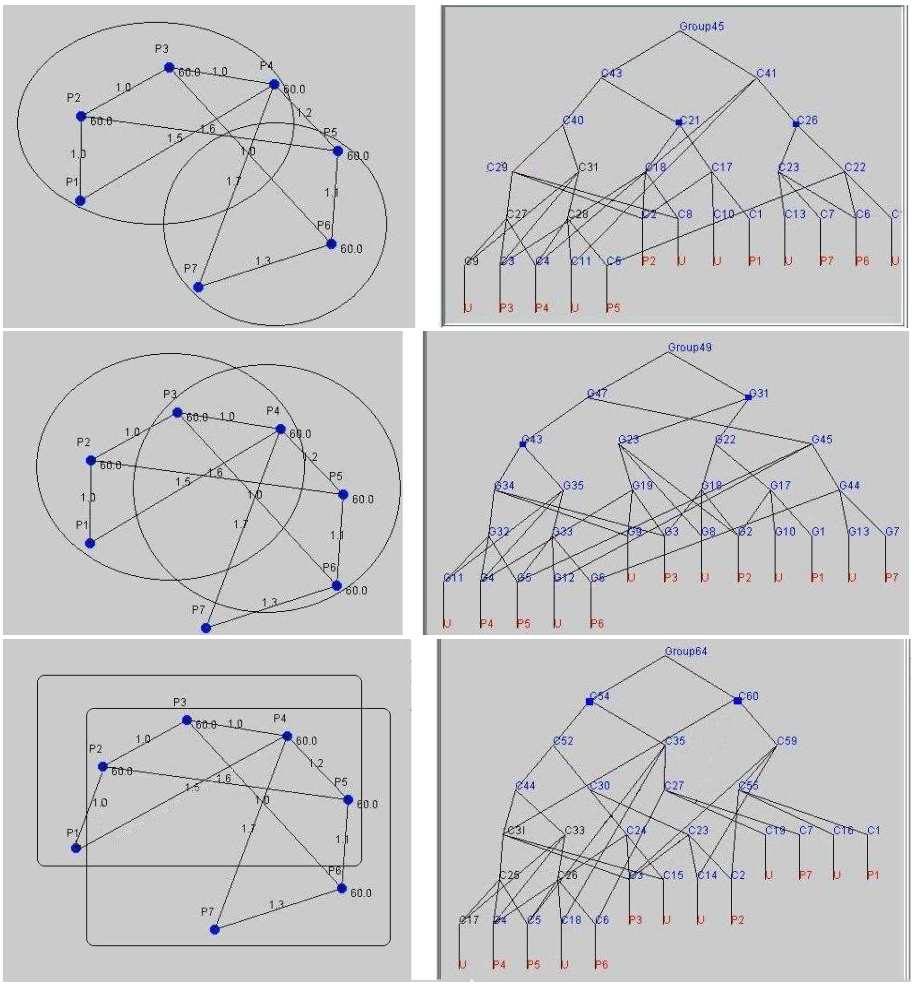

Figure 10: Left: input partial decompositions for 3D constraint system shown in Figure 7; groups are features or subassemblies. Right: 3 different DR plans incorporating corresponding input decomposition. Features appear as clusters or, if underconstrained, their complete set of maximal clusters. Features may (not) intersect on (non) trivial subgraphs. Top: left group appears as C21 and right as C28. Middle: left group appears as C31 and right group appears as C43. Bottom: left group appears as C60 and right group appears as C54.

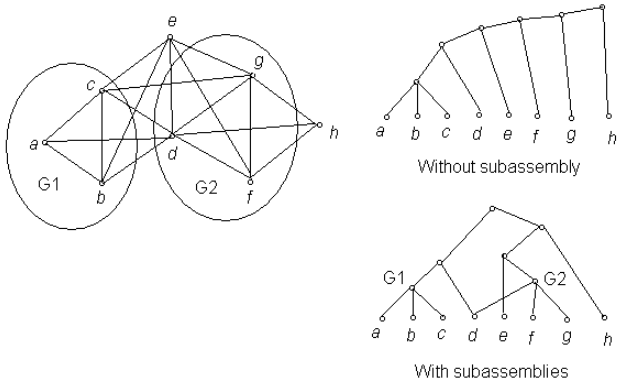

Figure 11: 3D constraint system of points and distance constraints, with input features or subassemblies G_1 and G_2. Two DR-plans are shown, one that incorporates the features and one that does not

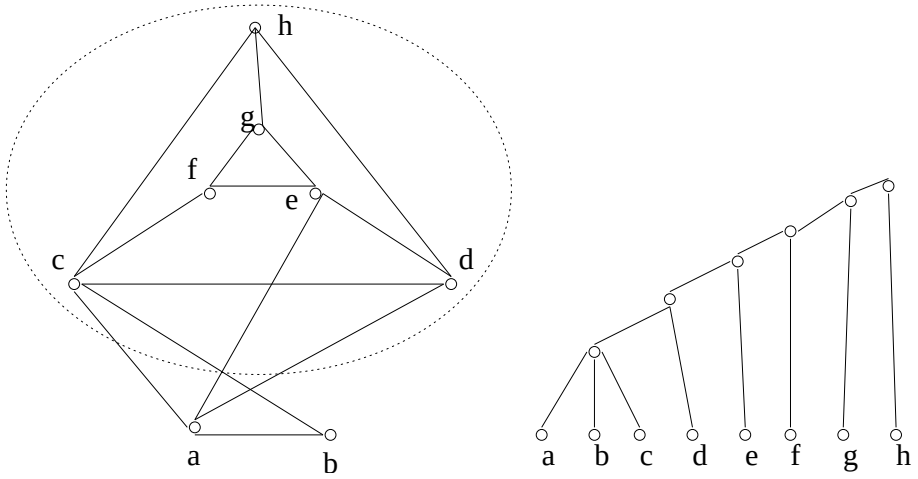

Figure 12: Triangle-decomposable 2D constraint system of points and distance constraints that is a counterexample to the Church-Rosser property and hence to the feature incorporation property of triangle decomposition based DR-planners. If a triangle in the circled part is first picked, DR-planner cannot continue. Right shows triangle decomposition starting from a different triangle.

generate DR-plans that *preserve these properties as much as possible*, given the restriction that these DR-plans are forced to incorporate a given input feature decomposition.

3.1 A related, different problem

The ability of a DR-planner to incorporate any input feature decomposition into a DR-plan of a constraint graph G implies that the DR-planner can find the clusters of the graph in any order that is consistent with containment. This is called the *Church Rosser* property of the DR-planner and has been investigated in [11, 12] and [23]. A formal definition is the following.

• Assume that a DR-planner P constructs its output DR-plan (a dag) for a constraint graph G bottom up, i.e., if a cluster C contains cluster D, and both appear in the dag output, then D has been found and inserted into the DR-plan before C. Except for this restriction, the DR-planner is allowed to find and insert any cluster of G into the output dag at any stage. The DR-planner P has the Church-Rosser property if any dag that it outputs is a valid DR-plan for the input graph G.

The Church Rosser property is weaker and does not imply the feature incorporation property, especially when the DR-planner in question does not perform a generalized analysis, but works only on graphs that have a restricted type of DR-plan.

For example, consider a DR-planner that requires the graph G to be decomposable into clusters that have a specific structure or topological "pattern" such as [11, 12], and only finds such clusters. The DR-planner could have a Church-Rosser property in that it would successfully find a DR-plan regardless of the order in which *such* clusters are found and processed. However, by simply picking a feature which is a cluster that in G that does not have that pattern, the DR-planner would not be able to incorporate that feature in its DR-plan and hence would not possess the feature incorporation property. Note that in the specific case of [11, 12], it can be shown that a graph is well-constrained and "triangle decomposable" if and only if *all* clusters in it are triangle-decomposable, (although this proof does not appear in [11, 12]). So in this case the Church-Rosser property implies the feature incorporation property; however in this case both properties do not hold for overconstrained graphs (see Figure 12); and as pointed out above, the presence of consistent overconstraints may be crucial in order to effectively mix feature-based and constraint-based representations.

Finally, even if a DR-planner has the Church-Rosser property and is in fact shown to be amenable to feature incorporation, the actual algorithmic problem still remains: of *efficiently* incorporating a given

input feature decomposition into a DR-plan while minimally altering its other desirable properties. This is the problem that we address here.

4 The Frontier Vertex Algorithm (FA) DR-Planner

Here we sketch the essential algorithmic details of the dof-rigid Frontier vertex (FA) DR-planner which are absolutely necessary to describe our feature incorporation algorithm. The dof-rigid FA DR-planner satisfies the properties discussed in the Section 2. The basic idea of this DR-planner and its performance was presented in [23]; a complete formal description along with analysis of correctness and (cubic) complexity and proof of the completeness property are given in [31], [32], [38]. A pseudocode is provided in [39] and the documented code can be found in [45]. A method of combinatorially obtaining an optimal, stable algebraic system for solving C, i.e., for obtaining optimal covering sets is given in [41, 44]. The method for obtaining all possible sets of reducible constraints for overconstrained clusters of dof-rigid Frontier vertex DR-plans, both for 2D and 3D, and modifying the DR-plan once they have been removed, is presented in [15]. Other desirable properties such as systematic correction of underconstraints, and amenability to efficient updates of geometric primitives or constraints are given in [38, 39].

4.1 The FA DR-planner structure and key methods

The *input* is the constraint graph G and the *output* is a DR-plan of quadratic width, satisfying the completeness property, with constant best-choice approximation factor.

```
Repeat
   pick a cluster C from clusterqueue CQ
   Distributecluster(C) in cluster graph $CG$
   if new cluster C' is found (containing C)
   then
     Complete (C'G)
              [recursive procedure builds complete sub DR-plan for cluster
               graph restricted to C', i.e., input constraint graph restricted
               to C', starting from those subclusters of C' that are already
               present in DR-plan; modifies DR-plan, inserting C' and
               new found subclusters of C' into it.]
     insert C' at the end of CQ
     Combine CQ (iterative procedure that modifies both DR-plan and CQ
                 until no further combining is possible)
     update CG by (frontier vertex) Simplify(clusters in CQ)

     remove C from CQ
Until CQ is empty

if DR-plan has more than 1 root, then Complete(CG)
```

We describe the methods: *DistributeCluster, Simplify, and Combine* to the extent necessary to describe and analyze our feature incorporation algorithm. *DistributeCluster* in turn relies on 3 methods based on network flow for locating dof-rigid clusters: *DistributeVertex, DistributeEdge and PushOutside*. It is not necessary to describe Complete since it is unaffected by the feature incorporation algorithm. For these, the reader is refered to [31, 32]. Briefly: and Complete recursively calls itself as well as Distributecluster, Combine and Simplify and requires care since completeness competes with the small width property of DR-plans as mentioned in Section 2.

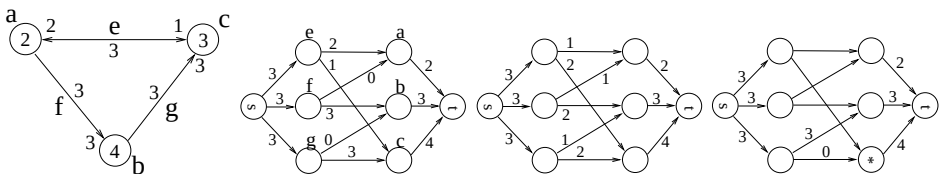

Figure 13: From Left. Constraint graph G with edge weight distribution. D is assumed to be 0 (system fixed in coordinate system); A corresponding flow in bipartite G^*. Another possible flow. Initial flow assignment that requires redistribution and search for an augmenting path

4.1.1 Distributing Edges

The dense subgraph isolation algorithm is used repeatedly to find dof-rigid clusters. It was first given in [19, 20] as a modified incremental network maximum flow algorithm. We assume standard working knowledge of network flow. The key routine is the *distribution* of an edge (see the DR-planner pseudocode in [39], and opensource code in [45]) 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." and called the *DistributeEdge* routine. 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 13, 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). The search for the augmenting path while distributing this edge marks the required dense graph. If the found subgraph is not overconstrained, then it is in fact minimal. If it is overconstrained, [19, 20] give an efficient algorithm to find a minimal cluster inside it.

4.1.2 Simplifying Clusters

We now describe the method *Simplify*. This frontier vertex simplification was given in [23, 21]. 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 - in which case, it lacks one dof). 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 14 illustrates this iterative simplification process ending in the final DR-plan of Figure 9.

The flows on the internal edges and the core vertex are inherited from the old flows on the internal edges and internal vertices. The undistributed weights on the internal edges simply disappear. The undistributed weights on the frontier edges are distributed (within the cluster) as much 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.

4.1.3 Datastructures

Four datastructures are maintained. The *cluster* datastructure for a cluster C contains data on the simplification of the cluster (frontier vertices, edges etc.), the original graph vertices and edges corresponding

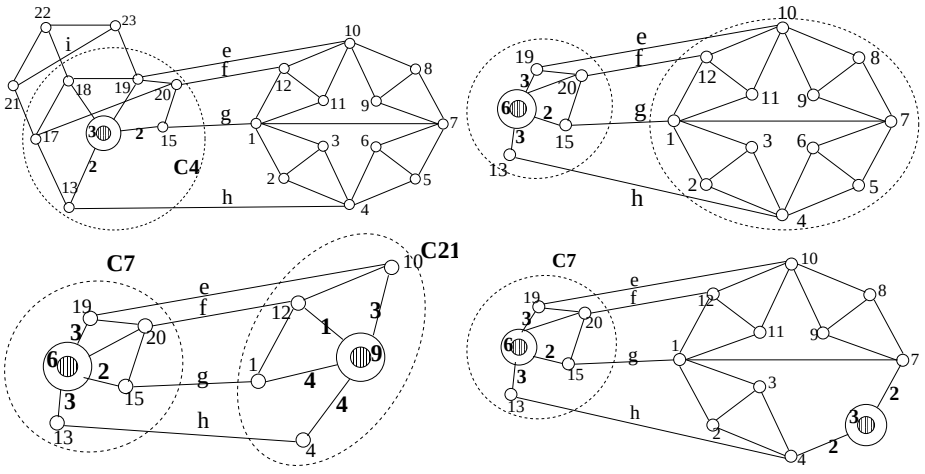

Figure 14: From left: FA's simplification of graph giving DR-plan in Figure 9; clusters are simplified in their numbered order: C4 is simplified before C7 etc.

to C, and pointers to the roots of the current sub DR-plan rooted at C (may or may not be complete sub-DR-plan depending on the stage of the algorithm). The *DR-plan* datastructure just contains pointers to the clusters that are currently the top-level clusters in the DR-plan. The *flow or cluster graph*, CG contains the current simplified graph where the found clusters have been simplified using the frontier vertex simplification. It also contains all the current flow information on the edges. a *cluster queue*, CQ which is the top-level clusters of the DR-plan that have not been considered so far, in the order that they were found.

Initialization. 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.

4.1.4 Distributing Vertices and Clusters

The method *DistributeVertex* distributes all edges (calls DistributeEdge) connecting the current vertex to all the vertices considered so far. When one of the edges cannot be distributed a minimal dense cluster C is isolated, as described in Section 4.1.1.

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 entire, i.e., its top level clusters are not maximal. The next level of clusters are found by distributing the (completed) 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. Then 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 is shown in [31, 32] 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.

4.1.5 Combining clusters

Next, we emphasize the parts of the algorithm that ensure a crucial property of the output DR-plan that the feature incorporation algorithm must deal with, namely small width, while maintaining complete decomposition of clusters (two types) defined in Section 2. The FA DR-planner controls the width of the FA DR-plan to ensure FA achieves a quadratic bound on DR-plan width by maintaining the following invariant of the clusterqueue *whenever DistributeCluster is called, every pair of clusters in the cluster*

queue and cluster graph intersect on at most a trivial subgraph (i.e., a subgraph which on resolving incidence constraints either represents to a single point in 2D or a variable or fixed length line segment in 3D). FA does this by repeatedly performing the *Combine* operation each time a new cluster is isolated.

The operation is to iteratively *combine* $N \cup D_1$ with any clusters $D_2, D_3, \ldots$ based on a nontrivial overlap. 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 and cluster queue after removing D_1, D_2, D_3 ....

5 Incorporating an input feature decomposition into an FA DR-plan

We develop a new algorithm that solves the algorithmic problem of Section 3. I.e., it permits the FA DR-planner ([31, 32], sketched in Section 4) to incorporate into its output DR-plan an input feature decomposition or conceptual, design decomposition.

5.1 More detailed requirements on the method

A recursive method called *DistributeGroup* drives the new FA DR-planner It is called on the root node of the input feature decomposition, which, by convention, represents the entire graph. Let $G_1, G_2, .., G_m$ be the children of some parent feature G in the input decomposition.

Now that the FA DR-planner's structure has been described, we can give a more detailed set of requirements on the feature incorporation algorithm based on the formal problem requirements of Section 3.

1. First of all, we need to ensure that each feature G_i itself appears in the output DR-plan provided it is a valid cluster. If it is not a cluster, then a complete maximal decomposition of it is obtained. This implies that a separate DR-plan for each G_i needs to be obtained and all of these DR-plans should appear within the DR-plan for G.

2. Secondly, while we cannot maintain the same width W as an FA DR-plan that is not required to incorporate any features (for the same constraint graph), we would like to have a width significantly smaller than $O(WF)$, where F is the number of features in the input feature hierarchy. Instead of specifying the desired complexity, we simply require the "best possible." I.e., we require that consistency between the DR-plans of the different G_i's needs to be established, in case they have shared objects. In particular, in the 3D constraint system in Figure 15, assume G_1's DR-plan has been obtained and has an intermediate cluster *bcd*. When the DR-plan for G_2 is obtained, the same cluster should appear.

3. Thirdly, for efficiency, we would like the new DR-planner - while working on some G_i - to use as much of the flow, cluster and DR-plan information that it already obtained while working on some earlier G_k, so that the entire complexity is proportional to the width or number of clusters in the final DR-plan.

Thus the key issues are: how are the flow or cluster graph (see description of FA in Section 4) and the output DR-plan efficiently maintained and modified as each G_i is worked on, so that the above requirements are satisfied.

Finally, note that since the input decomposition may be a *dag*, and not a tree, some children may have more than one "parent." However, *DistributeGroup* must be performed only once on any given group G; When another parent of G calls *DistributeGroup(G)* at a later point in time, the stored results for the flow or cluster graph and DR-plan for G should be returned.

5.2 Distributing Groups or Features

We discuss 3 distinct cases that need to be dealt with differently at the crucial steps of the algorithm.
Case 1: the children of G are mutually pairwise disjoint, i.e, do not overlap each other

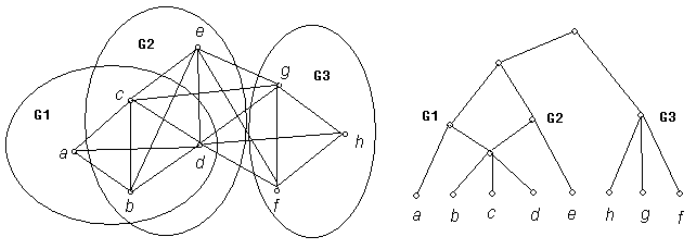

Figure 15: Consistency between the DR-plans of 2 groups of graph in Figure 11

Case 2: G_i's overlap with $G_k (1 \leq k \leq i-1)$ consists entirely of frontier vertices of the top level clusters in the merged DR-plan of the G_k's.
Case 3: for at least one of the $G'_k s (1 \leq k \leq i-1)$, G_i's overlap with G_k includes vertices (of the original graph) that map to the core of one of the clusters of G_k.

The method *DistributeGroup*(G). consists of 4 steps. First, for each child group G_i of G, it performs the following 3 steps.

Note. For clarity of exposition, we prefer to not give a pseudocode for *DistributeGroup*(G), but rather explain successively each of the 4 steps for all three of the cases. When we refer to the "old" DR-planner, we mean the pseudocode of Section 4; and the "new" refers to the old FA DR-planner augmented by the *DistributeGroup* driver. A detailed pseudocode of the entire FA DR-planner including the feature incorporation algorithm can be found in [39].

Step 1: The old FA DR-planner of Section 4 is called on G_i and starts a new DR-plan for G_i (which will eventually get merged with the DR-plans of the other children of G after Step 3 below). Then the new DR-planner uses different options for the flow or cluster graphs for the three cases.
Case 1: Use the current flow or cluster graph by freezing all the edges and vertices outside of G_i;
Case 2: Use the current flow or cluster graph, modify the flow on carefully selected edges and vertices outside of G_i, *mark*ing them and freezing them; I.e. for each edge e with one endpoint in G_i and another endpoint outside G_i, if there is any flow f on this edge towards G_i, then remove it and instead add f to the undistributed flow capacity on this edge e. Mark this edge e as having undistributed flow.
Case 3: Create a new local copy of a flow graph for G_i alone (which will eventually be used to update the current flow graph in Step 3 below).

Step 2: The DR-planner continues with a recursive call to *DistributeGroup(G_i)* during which it ensures that *DistributeEdge* is run on the undistributed flow on all marked edges within G_i. An edge e is un*mark*ed only if distribution is successful on all undistributed units on e.
Step 3: The DR-planner merges the DR-Plan of G_i with the DR-Plans of G_1 through G_{i-1}. This includes merging copies of clusters that could have been independently found by the DistributeGroup method on 2 different groups. It additionally includes putting clusters together to form larger clusters, based on amount of overlap; augmenting current flow graph (merged flow graphs so far) using the local flow or cluster graph for G_i. This latter part includes not only combining clusters but also modifying flows.

More specifically, the following merging operations are performed.

First, if a cluster in the DR-plans for $G_1, G_2, \ldots, G_{i-1}$ appears again in the DR-plan for G_i, the two copies are linked to prevent replication of effort during the solving stage. While the distinct groups of the input decomposition that are present in the sub-DR-plan of each cluster copy will have to be solved, the cluster itself and any cluster in its sub-DR-plan that is not a subset of a group in the input decomposition, will only have to be solved once. Note that merging the cluster copies (by taking the union of their parents and union of their children) will violate the so-called cluster minimality property of a good DR-plan (mentioned in Section 3), since a proper subset of the children would already form the cluster.

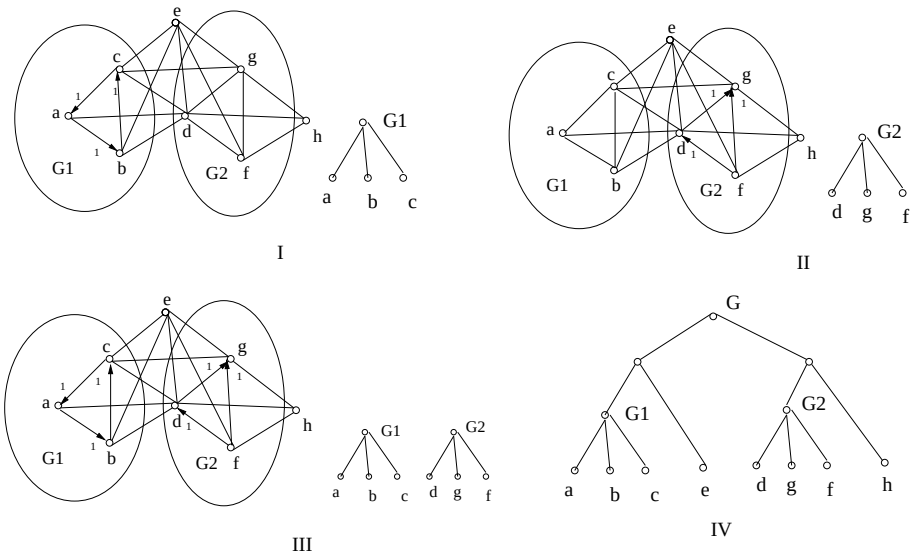

Figure 16: Case 1: Child Clusters do not overlap

Next, the DR-planner looks at the top level clusters of the merged DR-plan for $G_1, G_2, \ldots, G_{i-1}$ and the top level clusters of the DR-plan for G_i. If any pair of these say C and D intersect nontrivially, on more than 2 points in 2D and 3 points in 3D, then create a parent cluster $C \cup D$ of C and D, in the DR-plan; (this is the same as the *Combine* operation on clusters performed by the basic FA - Section 4), and making the corresponding simplification in the flow or cluster graph, described below.
Cases 1 and 2: Because the local flow graph is inherited from the flow graph for G, no additional modification is needed;
Case 3: removes all flow from the non-cluster edges that are in the overlapped part between G_i and that part of G that has been completed so far i.e: $G_1, \ldots, G_{i-1}$ and *mark*s their entire weight as undistributed. These edges will be redistributed when the clusters that contain them are distributed.

Step 4: Once the DR-plans of all the G_i's have been combined, the DR-planner proceeds as described in Section 4 on the resulting flow or cluster graph of G, performing *DistributeCluster* on the clusters in them, potentially isolating and simplifying new clusters that contain the G_i's, modifying the cluster queue and the DR-plan, until the DR-plan for G is completed.

We now describe how the above algorithm works on 3 examples that represent the 3 cases.

5.2.1 Example

For the 3D example of points and distance constraints in *Case 1* in Figure 16, Part I shows the flow graph and the cluster queue (see Section 4) after the DR-plan for G_1 has been constructed. When the old DR-planner starts to distribute G_2, it creates a new cluster queue for G_2 and inherits the flow graph in Step 1. After the old DR-planner in finished with G_2, in Step 2, the DR-plan of G_2 and the flow or cluster graph are shown in Part II. Then the new DR-planner tries to combine them in Step 3 and the results are shown in Part III of the figure. The final DR-plan of G which is obtained after Step 4 is shown in Part IV.
For the 3D example of points and distance constraints in *Case 2* in Figure 17, Part I shows the flow graph and cluster queue after the DR-plan for G_1 has been completed. When the DR-planner starts to distribute G_2, it creates a new cluster queue for G_2 and inherits the flow graph in Step 1. It also removes the flows on the edge *ce* and *cg* and *mark*s them. After the old DR-planner is finished with G_2 in Step 2, the DR-plan of G_2 and the flow graph are shown in Part II. Since G_1 and G_2 overlap on 3 points, the new DR-planner combines them in Step 3 and the results are shown in Part III. The final DR-plan of G which is obtained after Step 4 is shown in Part IV.

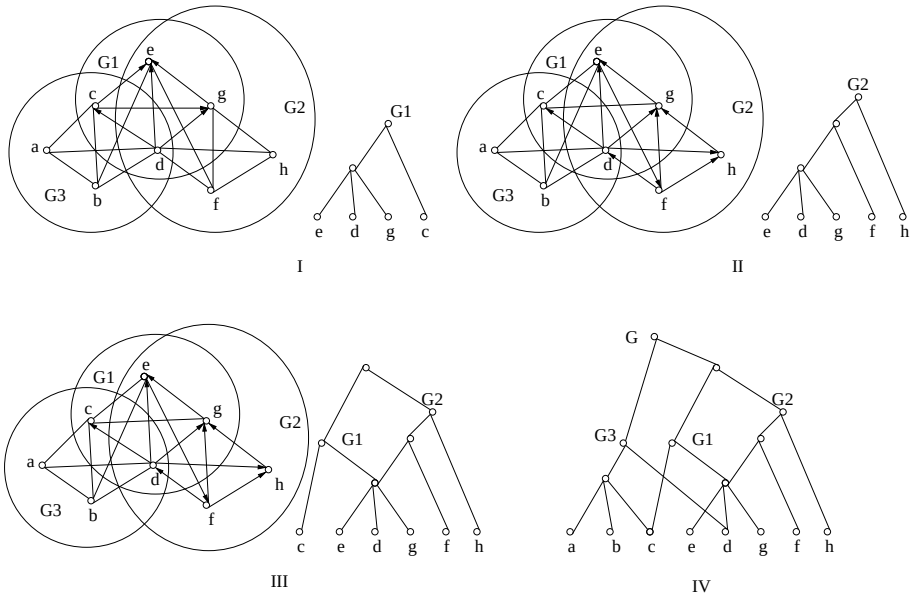

Figure 17: Case 2: input groups overlap on frontier vertices

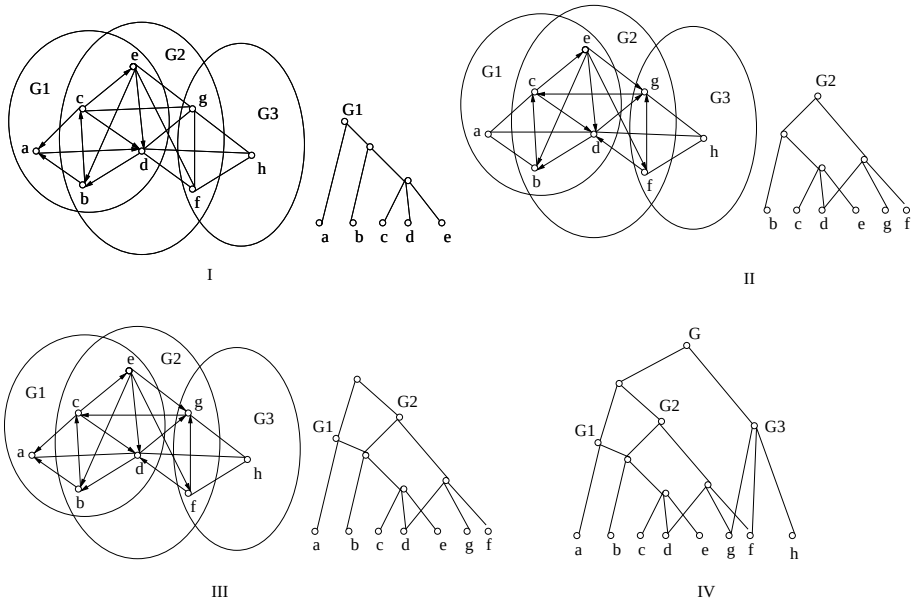

Figure 18: Case 3: the overlapped part includes non-frontier vertices

For the 3D example of points and distance constraints in *Case 3* in Figure 18, Part I shows the flow graph and cluster queue after the DR-plan for G_1 has been constructed. When the old DR-planner starts to distribute G_2, it creates a new cluster queue for G_2 and the flow graph in Step 1. After the old DR-planner is finished with G_2 in Step 2, the DR-plan of G_2 and the flow graph are shown in Part II. Since G_1 and G_2 overlap on 4 points, the DR-planner combines them in Step 3 and the results are shown in Part III. The final DR-plan of G which is obtained after Step 4 is shown in Part IV.

5.3 Proof of Correctness and Complexity

We show how the 3 requirements of Section 5.1 are met.

From Steps 1 and 2, we know that for each decomposition G_i, the algorithm creates a new DR-plan. This ensures, by the properties of the old FA DR-planner given in Section 4, that each feature G_i appears in the output DR-plan provided it is a valid cluster, and otherwise a complete decomposition into maximal clusters is obtained. In Step 3, the DR-planner checks the pair of the top level clusters of DR-plan for G_i and those of the combined DR-plan for $G_1, G_2, \ldots, G_{i-1}$ and combine them if possible. Because this process is executed for each G_i after its DR-plan is established, the algorithm combines the DR-plans for all the G_i's to give the DR-plan for G. Thus Requirement 1 is met.

In Step 3, the new algorithm links two copies of the same cluster in different G_i's. Thus, there is only one copy of this cluster in the DR-plan of G. So, the consistency between the DR-plans is ensured, satisfying Requirement 2.

Finally, the flow information in the 3 cases is either used as such, copied and restored, or is chalked up as undistributed units which will be re-distributed in Step 2 by the old DR-planner. This guarantees that flow information remains accurate throughout, given correctness of old DR-planner. Also, notice that for any given edge, the number of times it is re-distributed is no more than the number of groups or features in the feature hierarchy that share that edge. This ensures that any additional time spent (beyond that of the original DR-planner) is proportional to the number of features in the input feature hierarchy; thereby ensuring Requirement 3.

5.4 Preserving properties of the old FA DR-planner

Incorporation of features into the DR-plan leaves entirely unchanged many desirable properties of the the output DR-plan mentioned in Section 2, simply because the old DR-planner is called by the *Distribute-Groups* driver at each stage to actually construct the DR-plan. These include properties such as detection and characterization of over and under constrainedness [23, 15], completeness [31, 32], systematic correction of underconstrainedness by giving so-called *completion constraints* amenability to efficient updates (addition or deletion or modification) of geometric primitives or constraints [38, 39], or navigation of the solution space [40].

Next, we briefly discuss some desirable properties of the FA DR-plan that are affected by the feature incorporation algorithm as well as properties that are only relevant in the presence of features.

5.4.1 Complexity and Width

Let n is the number of vertices of the input constraint graph and k is the number of features in the input feature decomposition. Using the argument given in the proof of correctness and complexity, and the complexity of the old DR-planner, the following hold. If all the features are either disjoint or contain one another (Case 1). new algorithm's time complexity is $O(n^3)$, width $O(n^2)$ (the complexity of the old DR-planner). If the features intersect on trivial subgraphs or contain one another (Case 2), the complexity is $O(n^3+k)$, width $O(n^2+k)$. Finally if the features could intersect on nontrivial subgraphs, the best bound on complexity is $O(n^3k)$, width $O(n^2k)$ (we omit a finer, but significantly more cumbersome complexity expression in terms of sizes of the intersections etc.) These are the best complexities one can expect. The first factor is the complexity of the underlying old FA DR-planner and in typical cases, the second factor is insignificant.

5.4.2 Optimality

Concerning optimality, the FA DR-planner's best choice approximation factor is unaffected by the new augmentation. The proof is the same as for the old DR-planner. [23] *Among all DR-plans that incorporate the given input feature decomposition*, the FA DR-planner augmented by the feature incorporation algorithm can find one whose maximum fan-in cluster has a fan-in that is atmost a constant factor larger than the optimum.

Also, as mentioned earlier, the feature incorporation does not affect he completeness property, so the algorithms of [41] and [44] can still be used to find an optimal covering set.

5.4.3 Correction of Overconstraints

Another property of FA DR-plans that is superficially affected by the presence of features is the systematic correction of overconstraints, i.e., the method presented in [15]. Clearly feature incorporation does not affect the ability to detect overconstraints and isolate of a complete set of overconstraints that can be removed without making the *entire* graph underconstrained. However, correction of overconstraints typically results in removing some clusters in the DR-plan, since they become underconstrained, although the entire graph remains well-constrained. In the presence of features, it is reasonable to require that *no* feature that was previously a cluster is made underconstrained by the correction, i.e., the set of so-called *reducible* overconstraints is smaller. However, the overconstraint correction method of [15] explicitly provides a list of reducible overconstraints *directly associated* with each cluster in the DR-plan. See Section 2 for definitions. Hence, the required modification is straightforward: the new set of reducible overconstraints that preserve features is the union of all the reducible sets of overconstraints directly associated with each cluster feature in the DR-plan, together with the union of all the reducible sets of overconstraints for clusters that are not descendants of any cluster feature in the DR-plan.

5.4.4 Updating the Feature hierarchy

Finally, a property listed under the output requirements of the feature incorporation problem in Section 3 is the ability to update the input feature decomposition and correspondingly efficiently update the DR-plan.

Removal of a feature is straightforward. If the feature is a cluster it simply entails the removal of the corresponding node C from the DR-plan, and all of its descendants that are inessential children of their other parents who are not descendants of C (see Section 4 for definition of essential clusters). If the feature is not a cluster, then all of its maximal proper clusters are present as nodes in the DR-plan and these are treated like C above. The DR-planner does not need to be involved in this simple edit of the DR-plan.

Addition of a feature is more involved. There are two cases. In the case where the feature is not contained within an existing cluster of the DR-plan (it could be contained in the single root if the graph is wellconstrained), then the addition of the feature is straightforward since it will enter the upper most level of the DR-plan. It is simply treated by the Distributegroups method as though it is a (last) child of the root of the input feature decomposition. If the feature contains new geometric elements and constraints, these are processed using the update method for FA DR-plans, given in [38, 39]. In case the new feature F is contained within one or more of the existing clusters C_i in the DR-plan, it is first assumed to be a cluster and inserted in the DR-plan as a child of the C_i and as a parent of all the maximal clusters D_i that are contained in F and are present in the sub-DR-plan rooted at any of the C_i's. Since F lies inside an already processed cluster, no flow information is available. A cluster or flow graph of F is created by using the frontier vertex simplifications of these maximal clusters. These frontier vertices are connected using edges from the original graph. A cluster queue with these clusters is created and these clusters are treated as the top level of the DR-plan for F constructed so far. I.e., Steps 3 and 4 of the *DistributeGroup* method on F are executed, taking the sub-DR-plan rooted at F as the input feature decomposition and assuming that *DistributeGroup* has already been called on the children of F in this feature decomposition. During Step 4, since none of the edges in the flow or cluster graph constructed for F have been distributed, *Pushoutside* and other methods cannot assume that the edges in the cluster graph for F have been distributed, *DistributeEdge* is run again on these edges.

5.4.5 Implementation

A detailed pseudocode that includes the new feature incorporation algorithm can be found in [39]. Documented opensource code can be downloaded at [45] (use post-December-2003 versions for 3D) To use the feature incorporation option after opening the main sketcher window, and after pulling up (or drawing) a sketch: press 'ctrl' key and left click the objects which should be in the same feature. You will see the color of all selected objects is changed. Tip: you can use left mouse button to draw a rectangle to select objects quickly, then use 'ctrl' + left click to modify selected set. ('ctrl' + left click on a selected object would unselect it.) Click 'Design' menu, then click 'Make new tree' to create each feature hierarchy (independent feature hierarchies for the same constraint graph provided by different "users" or multiple views, are combined to form a single composite feature hierarchy, internally). Then click 'Make new group' to create the feature. For each features, you could simply select the primitive objects in them and click 'Make new group'. You can check the features by clicking the 'group' tab in right-bottom of the window.

Note. Recall from Section 2 that the dof-rigid FA DR planner considered here does not deal with implicit constraint dependences. However, the more general, module-rigid FA DR-planner [42] deals with all known types of constraint dependences such as bananas and hinges. While incorporation of a feature hierarchy into the the module-rigid FA DR-planner [42] has been implemented in FRONTIER [33, 45], its description and analysis are beyond our current scope. We would like to note that the detection of module-rigidity crucially relies on the completeness of the underlying dof-rigid DR-planner, which is unchanged by the feature incorporation algorithm. More significantly, the notion of a module-rigid cluster includes so-called *dependent* clusters that are not self-contained, but they need to be resolved after others, imposing a *solving priority* order. DR-planners that can deal with such clusters have an edge in incorporating those features - as in procedural history based representations - whose very definition is based on previously defined features.

References

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

[2] V. Allada and S. Anand. Feature-based modeling approaches for integrated manufacturing: state-of-the-art survey and future resear ch directions. *International Journal for Computer Integrated Manugfacturing*, 8:411–440, 1995.

[3] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige. A geometric constraint solver. *Computer Aided Design*, 27:487–501, 1995.

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

[5] G. Brunetti and B. Golob. A feature based approach towards an integrated product model including conceptual design information. *Computer Aided Design*, 32:877–887, 2000.

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

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

[8] K.J. de Kraker, M. Dohmen, and W.F. Bronsvoort. Maintaining multiple views in feature modeling. In *ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 123–130. ACM press, 1997.

[9] U. Doering and B. Bruderlin. A declarative modeling system. In P. Brunet, editor, *CAD Systems Development -Tools and Methods*, page To appear. SpringerVerlag, 1999.

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

[11] I. Fudos and C. M. Hoffmann. Correctness proof of a geometric constraint solver. *Intl. J. of Computational Geometry and Applications*, 6:405–420, 1996.

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

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

[14] J.H. Han and A.A.G. Requicha. Modeler-independent feature recognition in a distributed environment. *Computer Aided Design*, 30:453–463, 1998.

[15] C Hoffman, M Sitharam, and B Yuan. Making constraint solvers more useable: the overconstraint problem. *CAD*, 36(4):377–399, 2004.

[16] C. M. Hoffmann and R. Joan-Arinyo. Distributed maintanence of multiple product views. *Manuscript*, 1998.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

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

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

[24] Christoph M. Hoffmann and Pamela J. Vermeer. Geometric constraint solving in R^2 and R^3. In D. Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry*. World Scientific Publishing, 1994. second edition.

[25] Christoph M. Hoffmann and Pamela J. Vermeer. A spatial constraint problem. In *Workshop on Computational Kinematics*, France, 1995. INRIA Sophia-Antipolis.

[26] R. Klein. Geometry and feature representation for an integration with knowledge based systems. In *Geometric modeling and CAD*. Chapman-Hall, 1996.

[27] R. Klein. The role of constraints in geometric modeling. In Bruderlin and Roller ed.s, editors, *Geometric constraint solving and applications*. Springer-Verlag, 1998.

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

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

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

[31] Andrew Lomonosov. Graph and Combinatorial Analysis for Geometric Constraint Graphs. Technical report, Ph.D thesis, Univ. of Florida, Gainesville, Dept. of Computer and Information Science, Gainesville, FL, 32611-6120, USA, 2004.

[32] Andrew Lomonosov and Meera Sitharam. Graph algorithms for geometric constraint solving. In *submitted, available upon request*, 2004.

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

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

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

[36] 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.

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

[38] M Sitharam. Graph based geometric constraint solving: problems, progress and directions. In Dutta, Janardhan, and Smid, editors, *AMS-DIMACS volume on Computer Aided Design (to appear), www.cise.ufl.edu/∼sitharam/dimacs.pdf*, 2004.

[39] M Sitharam. Frontier, an opensource 3d geometric constraint solver: algorithms and architecture. In *www.cise.ufl.edu/∼sitharam/partone.pdf, www.cise.ufl.edu/∼sitharam/partwo.pdf*, 2005.

[40] M Sitharam, A Arbree, Y Zhou, and N Kohareswaran. Solution management and navigation for 3d geometric constraint systems. *submitted, www.cise.ufl.edu/∼sitharam/esm.pdf*, 2004.

[41] M Sitharam, J Peters, and Y Zhou. Solving minimal, wellconstrained, 3d geometric constraint systems: combinatorial optimization of algebraic complexity. *Automated Deduction in Geometry (ADG), www.cise.ufl.edu/∼sitharam/skeleton.pdf*, 2004.

[42] M Sitharam and Y Zhou. A tractable, approximate, combinatorial 3d rigidity characterization. *Automated Deduction in Geometry (ADG), www.cise.ufl.edu/∼sitharam/module.pdf*, 2004.

[43] M. Sitharam and Y. Zhou. Characterization of rigidity for 2d angle and incidence constraints. *Manuscript; available upon request*, 2005.

[44] M. Sitharam and Y. Zhou. Determining an independent set of overlap constraints between rigid bodies. *Manuscript; available upon request*, 2005.

[45] 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.

[46] W. Whiteley. Rigidity and scene analysis. In *Handbook of Discrete and Computational Geometry*, pages 893 –916. CRC Press, 1997.