COMBINATORIAL DECOMPOSITION, GENERIC INDEPENDENCE AND ALGEBRAIC COMPLEXITY OF GEOMETRIC CONSTRAINTS SYSTEMS: APPLICATIONS IN BIOLOGY AND ENGINEERING

By

YONG ZHOU

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2006

I dedicate this work to my family.

ACKNOWLEDGMENTS

Thanks to all for their help and guidance.

TABLE OF CONTENTS

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

COMBINATORIAL DECOMPOSITION, GENERIC INDEPENDENCE AND ALGEBRAIC COMPLEXITY OF GEOMETRIC CONSTRAINTS SYSTEMS: APPLICATIONS IN BIOLOGY AND ENGINEERING

By

Yong Zhou

December 2006

Chair: Meera Sitharam
Major Department: Computer and Information Science and Engineering

This dissertation contributes to 7 problems in geometric constraints solving and applications. 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 give a combinatorial *approximate* characterization of such graphs which we call *module-rigidity*, which can be determined by a polynomial time algorithm. The new method has been implemented in the FRONTIER and has many useful properties and practical applications. We propose a combinatorial characterization conjecture of 2D rigidity of angle constraint system. We give the formal statement of generically independence. We are trying to prove it use induction on the number of points. Three of four cases have been proved. We also study the well-documented problem of systematically navigating the potentially exponentially many roots or realizations of well-constrained, variational geometric constraint systems. We give a scalable method called the ESM or Equation and Solution Manager that can be used both

for automatic searches and visual, user-driven searches for desired realizations. We also show that, especially for 3D geometric constraint systems, a further optimization of the algebraic complexity of the subsystems is both possible, and often necessary to solve the well-constrained systems selected by the DR-plan. We give an efficient algorithm to optimize the algebraic complexity of the wellformed system that is constructed by the algorithm given by Sitharam. we implement a randomized algorithm to compute one measure of the probability of the pathways and prospect to incorporate another probability measure, that will be obtained combinatorially by extending Hendricksons Theorem on rigidity circuits and unique graph realization into this algorithm.

CHAPTER 1
INTRODUCTION

Geometric constraint systems have been studied in the context of variational constraint solving in CAD for nearly 2 decades [1, 2, 3, 4].

A *geometric constraint system* consists of a finite set of primitive geometric objects such as points, lines, planes, conics and so on and a finite set of geometric constraints between them such as distance, angle, incidence and so on. 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 related to the FRONTIER geometric constraint solver [1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15], to the best of our knowledge, work on stand-alone 3D geometric constraint solvers is relatively sparse [16, 17].

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; that is, solutions that cannot be infinitesimally flexed to give another

nearby solution: the solution space (modulo rigid body transformations such as rotations and translations) consists of isolated points which are 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.

1.1 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 and so on,), that hold *generically*, that is, for all generic values for the constraint parameters (for example, for almost all distance values, in the case of distance constraint systems).

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 1–1, 1–2 and 1–5 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 1–3 1–4.

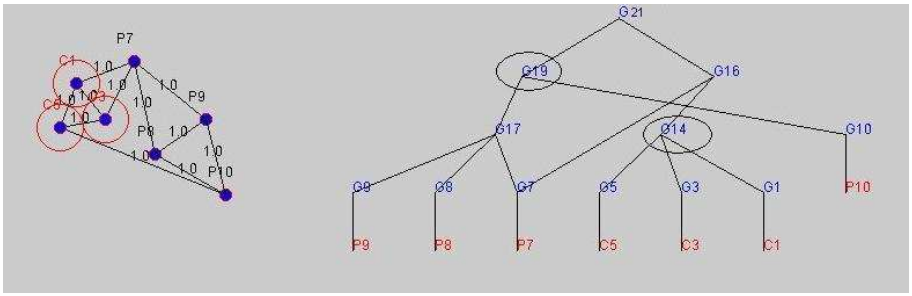

Figure 1–1: One 2D constraint system example.

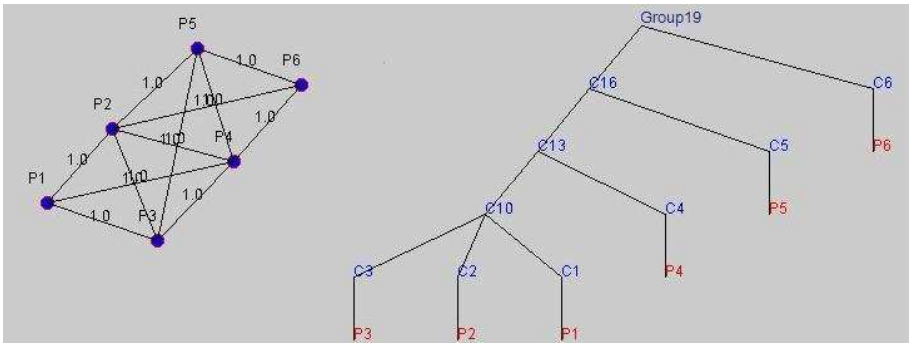

Figure 1–2: Simple 3D example drawn on 2D canvas with points and its DR-plan

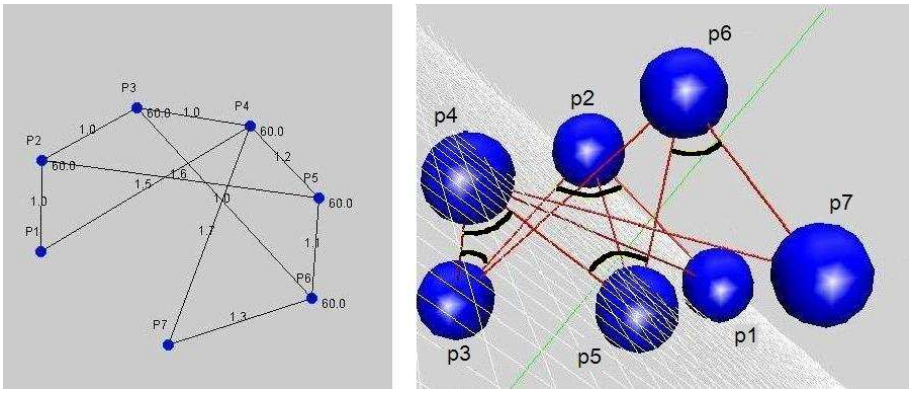

Figure 1–3: One 3D example drawn on 2D canvas with solution.

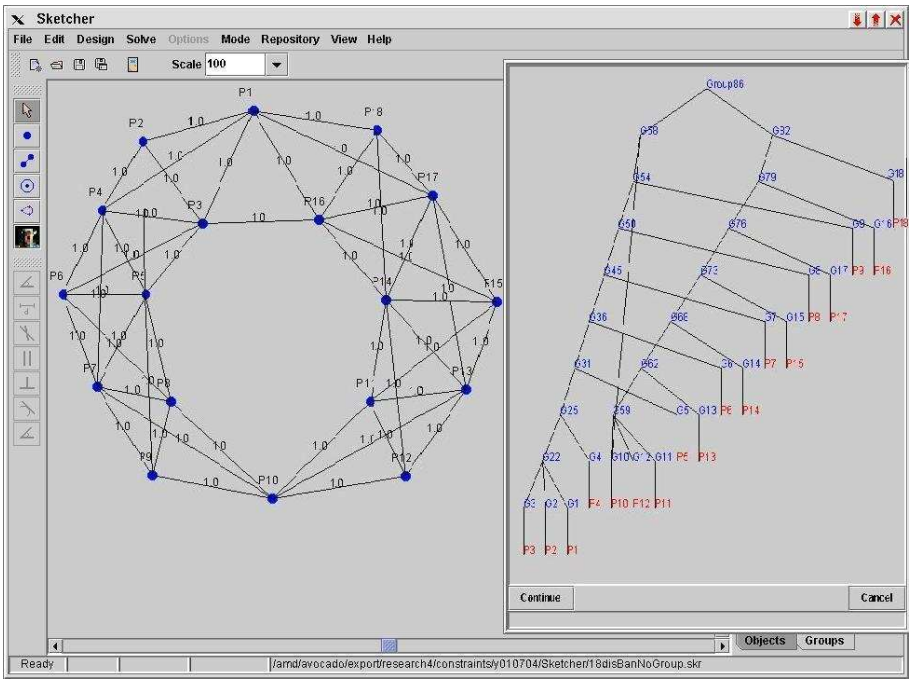

Figure 1–4: Geometric constraint system in 3D with "bananas" type

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).

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 at least 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 [18], 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*. One example (Figure 1–4) with 3D points and distance constraints illustrates the so-called "bananas" problem [19], which generalizes to the so-called "hinge" problem [20, 21]. 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 [19, 22], although several conjectures exist. There are no 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 [23]. For 3D points and distances, the notion of *module-rigid clusters* [12] (an extension of dof-rigid clusters defined above) deals with all aspects of the bananas and hinge problems, and so on, it correctly characterizes generic rigidity in all known cases. Currently, no counterexamples are known - of module-rigid constraint graphs that are not generically rigid.

1.2 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 Hoffmann et al. and Lomonosov [6, 24] 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 (Figures 1–5, 1–1, 1–2, 1–3, 1–4).

1.2.1 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. That is, 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

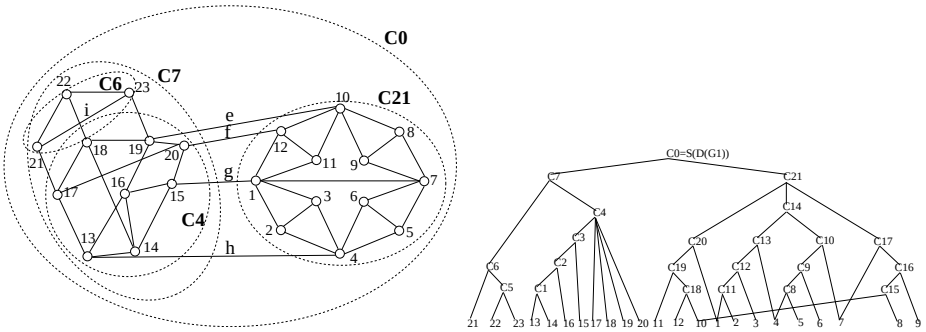

Figure 1–5: Constraint graph G1 and DR-plan in 2D.

should additionally admit an efficient method of removing overconstraints by making the appropriate changes to the DR-plan.

1.2.2 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. The problem of finding the optimal DR-plan of even a 2D distance constraint graph is NP-hard [9, 24], 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.

1.2.3 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, that is, 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.

1.2.4 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 underconstrained systems, and amenability to efficient updates of geometric primitives or constraints.

CHAPTER 2
PROBLEMS

2.1 Geometric Constraints System Within Feature Hierarchies

One application of geometric constraints system is mechanical computer aided design. However, a bottleneck in using geometric constraints system is that designers use another representation based on the conceptual hierarchy of features in a part or assembly.

In Chapter 3, we study the problem of enabling general 2D and 3D variational constraint representation to be used in conjunction with the 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.

2.2 Tractable Combinatorial 3D rigidity characterization

One big open problem is the question of rigidity of 3D constraint systems even when only distances are involved. The problem is that most combinatorial characterizations of rigidity will miss hidden dependencies between constraints and will be wrongly classify many classical nonrigid systems as rigid.

In Chapter 4, we give a polynomial time characterization called *module-rigidity* that is not fooled by any known such counterexample of nonrigid systems. 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" [19] 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 [9] 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 [10, 11, 15], [2] opensource 3D geometric constraint solver and has many useful properties and practical applications [13, 14, 25, 26, 27]. 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.

2.3 A 2D Rigidity Classification Conjecture

Laman's theorem shows one combinatorial characterization of 2D distance constraint system [18]. When other constraints are involved, *for example, angles*, there is no combinatorial characterization. In Chapter 5, we give a combinatorial characterization conjecture of 2D rigidity of angle constraint system. First, we give the formal statement of generically independence. We observe that for an given angle constraint system G in $2D$ with k dof, there is at least one point involved in

at most 7 angles. We divide it into 4 cases based on there exists at least one point with 0, 1, 2 or more angles involved. We are trying to prove it use induction on the number of points. The first three cases have been proved.

2.4 Solution Space Navigation for Geometric Constraint Systems

In Chapter 6, we study the well-documented problem of systematically navigating the potentially exponentially many roots or realizations of well-constrained, variational geometric constraint systems. We give a scalable method called the ESM or Equation and Solution Manager that can be used both for automatic searches and visual, user-driven searches for desired realizations. The method incrementally assembles the desired solution of the entire system and avoids combinatorial explosion, by offering the user a visual walkthrough of the solutions to recursively constructed subsystems and by permitting the user to make gradual, adaptive solution choices.

We isolate requirements on companion methods that permit (a) incorporation of many existing approaches to solution space steering or navigation into the ESM; and (b) integration of the ESM into a standard geometric constraint solver architecture. We address the latter challenge and explain how the integration is achieved. And we clarify these requirements essential and desirable for efficient, meaningful solution space navigation. Also, we sketch the ESM implementation as part of an opensource, 2D and 3D geometric constraint solver FRONTIER developed by our group.

2.5 Combinatorial Optimization of Algebraic Complexity

In Chapter 7 we show that, Most major geometric constraint solvers use a combinatorial or graph algorithm to generate a decomposition-recombination (DR) plan. A DR plan recursively decomposes the system of polynomial equations into small, generically rigid subsystems that are more likely to be solved by algebraic-numeric solvers.

Especially for 3D geometric constraint systems, a further optimization of the algebraic complexity of these subsystems is both possible, and often necessary to solve the well-constrained systems selected by the DR-plan. In Chapter 7, to attack this apparently undocumented challenge, we use principles of rigid body transformation and quaternion forms and we combinatorially optimize a function over the minimum spanning trees of a graph generated from DR-plan information. This approach follows an interesting connection between the algebraic complexity of a well-constrained system and the topology of its maximal well-constrained proper subsystems. The optimization has two secondary advantages: for navigating the solution space of the constraint system and for mapping solution paths in the configuration spaces of the subsystems.

We compare the reduction in algebraic complexity of the subsystem after optimization with that of the unoptimized subsystem and illustrate the practical benefit with natural examples that could only be solved after optimization.

2.6 Optimization of Wellformed Systems of Incidences

For tractability, many modern geometric constraint solvers recursively decompose an input geometric constraint system into standard collections of smaller, generically rigid subsystems or clusters. These are recursively solved and their solutions or realizations are recombined to give the solution or realization of the input constraint system.

The recombination of a standard collection of solved clusters typically reduces to positioning and orienting the rigid realizations of the clusters with respect to each other, subject to incidence constraints representing primitive, shared objects between the clusters and distance constraints.

The paper [28] shows that even for generically wellconstrained systems in 3D, finding a wellconstrained system of incidence constraints for recombining its cluster decomposition is a significant hurdle faced by geometric constraint

solvers. In general, we would like a wellformed system of incidences that preserves the classification of the original, undecomposed system as a well, under or overconstrained system. The author formally states and gives an efficient, greedy algorithm to find such a wellformed system.

2.7 Enumeration of Pathways for Macromolecular Structures

We consider the problem of explicitly enumerating and counting the assembly pathways by which an icosahedral viral shell forms from identical constituent protein monomers. This poorly understood assembly process is a remarkable example of symmetric macromolecular self-assembly occuring in nature and possesses many features that are desirable while engineering self-assembly at the nanoscale.

Sitharam and Mckenna [25, 29] give the new model of that employs a static geometric constraint graph to represent the driving (weak) forces that cause a viral shell to assemble and hold it together. The model was developed to answer focused questions about the structural properties of the most probable types of successful assembly pathways. Specifically, the model reduces the study of pathway types and their probabilities to the study of the orbits of the automorphism group of the underlying geometric constraint graph, acting on the set of pathways. The authors give a randomized algorithm to compute one measure of the probability of these pathways by faithfully sampling them.

In Chapter 9, we give the implementation of the algorithm and propose to incorporate another probability measure, that will be obtained combinatorially by extending Hendricksons Theorem [30] on rigidity circuits and unique graph realization into this algorithm. This involves both the theory of geometric constraints system and implementation of the new algorithm.

CHAPTER 3
GEOMETRIC SYSTEM WITHIN FEATURE HIERARCHIES

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, [31, 32] 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 maintenance 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 chapter, we denote such representations as *mixed* representations (Figure 3–2).

Previous work on such mixed representations can be classified into two broad types. The first type [33, 34], dictates a unified representation language which is an amalgamation of variational constraints with other representation languages such as CSG and Brep. The second type [35] 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.

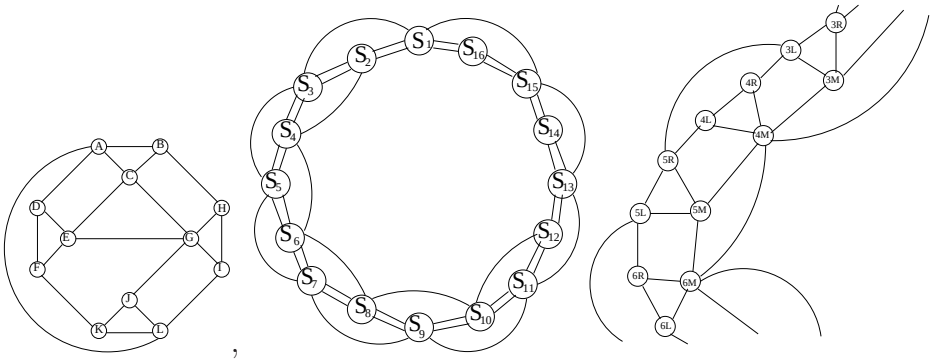

Figure 3–1: Small 2D constraint graph that is not triangle decomposable.

Our approach in this chapter 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, constraints between features at that level could be specified between primitive geometric objects or *handles* belonging to the features (Figure 3–2).

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 1.2) 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 (Figure 3–1).

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 [13], optimizing the algebraic complexity of subsystems sent to an algebraic-numeric solver [14], for dealing with explicit inconsistencies (overconstrained systems) [8], implicit or geometric redundancies and constraint dependences [12, 23], ambiguities (underconstrained systems) and for efficient updates and online solving [2, 11].

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* (Figures 3–2, 3–3, 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 1.2. Hoffmann et al. [6] first formalized the concept of a DR-plan as well as several performance measures of DR-planners some of which are relevant to this. It additionally gives a table of comparisons which shows that many of the previous DR-planners for example [3, 36, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45], (or any obvious modifications of them) would inherently fail to incorporate even tree-like input design decompositions or

feature hierarchies. We explain in Section 3.1 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 [13] 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 maintenance of multiple product views [46, 47], 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 (Figure 3–4). 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.

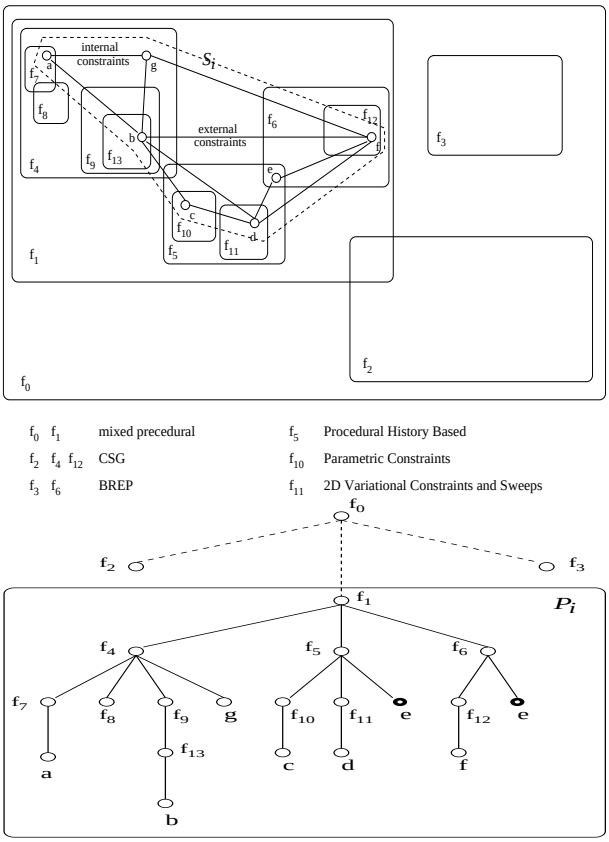

Figure 3–2: Constraint system S_i and underlying feature hierarchy P_i.

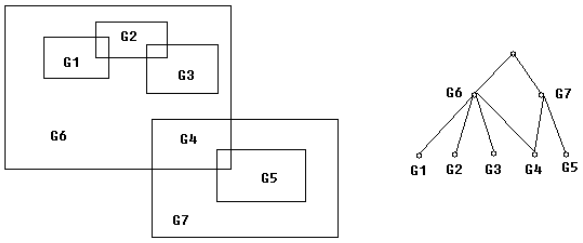

Figure 3–3: Feature hierarchy and directed acyclic graph.

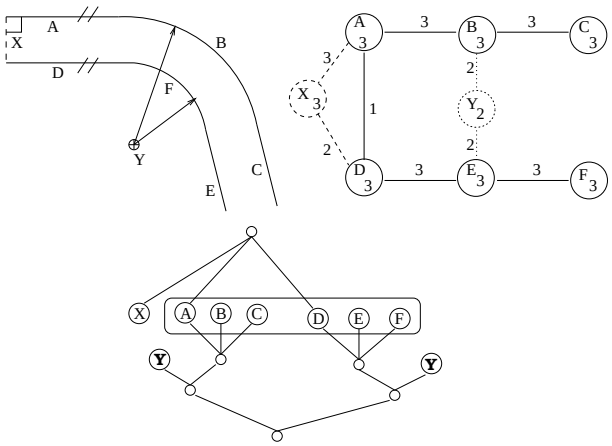

Figure 3–4: Constraint system and constraint graph showing multiple views.

3.1 Formal Problem Statement

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

Input: A 2D or 3D geometric constraint system (Section 1.1), S. A *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 (Figures 3–5, 3–3, 3–6).

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 1.2 must be preserved by the feature incorporation algorithm and its output DR-plan. Some of these properties require straightforward re-definition for feature

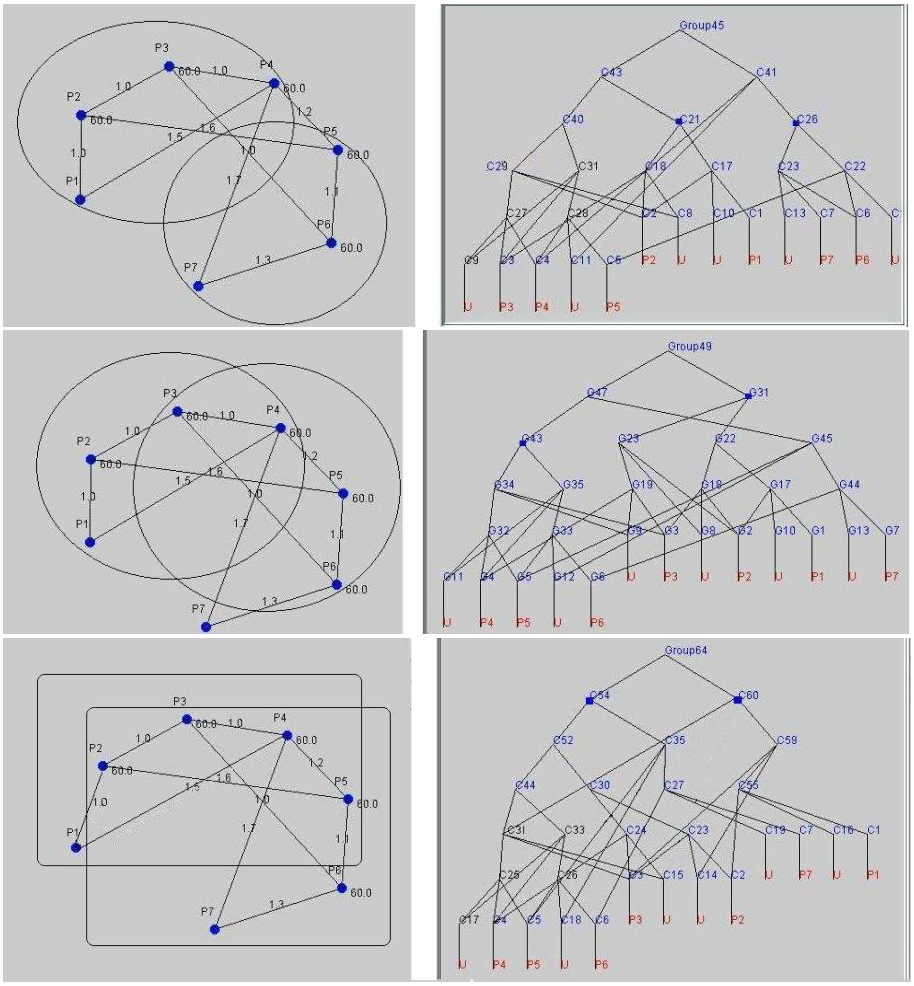

Figure 3–5: Partial decompositions for 3D constraint system (Figure 1–3).

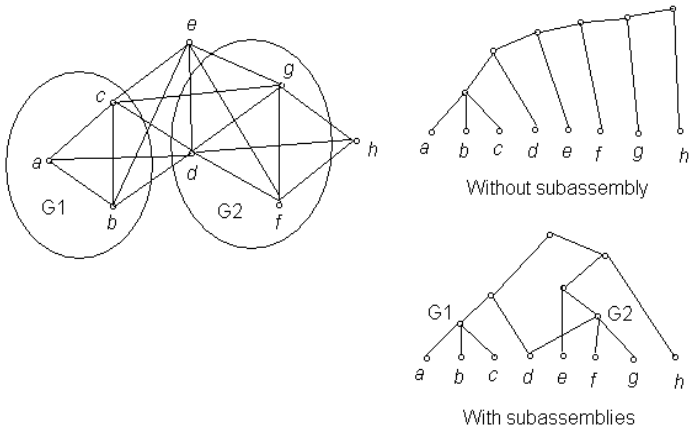

Figure 3–6: Two DR-plans of 3D example with input features.

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 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.

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 [6, 41, 42]. 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, that is, 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" [41, 42], and only finds such clusters. The DR-planner could have a Church-Rosser

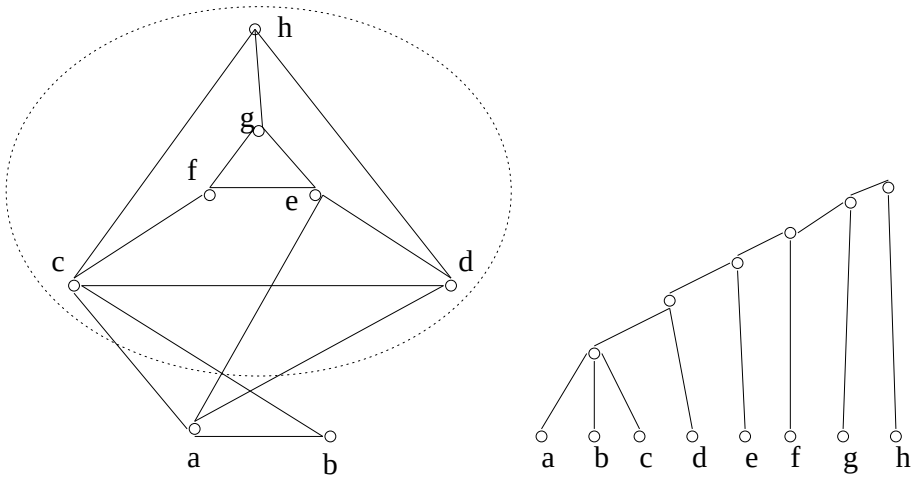

Figure 3–7: Triangle-decomposable 2D counterexample of Church-Rosser property.

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 Fuddos and Hoffmann [41, 42], 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 the papers of Fudos and Hoffmann [41, 42]). 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 (Figure 3–7); 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.

3.2 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 1.2. The basic idea of this DR-planner and its performance was presented by Hoffmann et al. [6]; a complete formal description along with analysis of correctness and (cubic) complexity and proof of the completeness property are given by Lomonosov and Sitharam [2, 9, 24]. A pseudocode is provided by Sitharam [11] and the documented code can be found on her CISE and GNU websites [15]. A method of combinatorially obtaining an optimal, stable algebraic system for solving C, that is, for obtaining optimal covering sets is given by Sitharam et al. and Sitharam and Zhou [14, 48]. 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 by Hoffmann et al. [8]. Other desirable properties such as systematic correction of underconstraints, and amenability to efficient updates of geometric primitives or constraints are given by Sitharam [2, 11].

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', that is, 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 Lomonosov and Sitharam [9, 24]. 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 1.2.

3.2.1 Distributing Edges

The dense subgraph isolation algorithm is used repeatedly to find dof-rigid clusters. It was first given by Hoffmann et al. [1, 5] 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 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 3–8, we may need to redistribute (find an augmenting path).

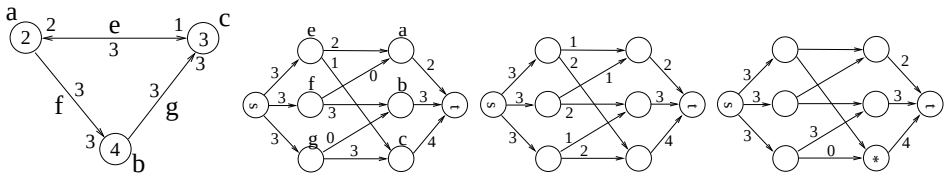

Figure 3–8: Constraint graph G with edge weight distribution.

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, Hoffmann et al. [1, 5] give an efficient algorithm to find a minimal cluster inside it.

3.2.2 Simplifying Clusters

We now describe the method *Simplify*. This frontier vertex simplification was given by Hoffmann et al. [6, 7]. The found cluster C interacts with the rest of the constraint graph through its *frontier vertices*; that is, 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 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).

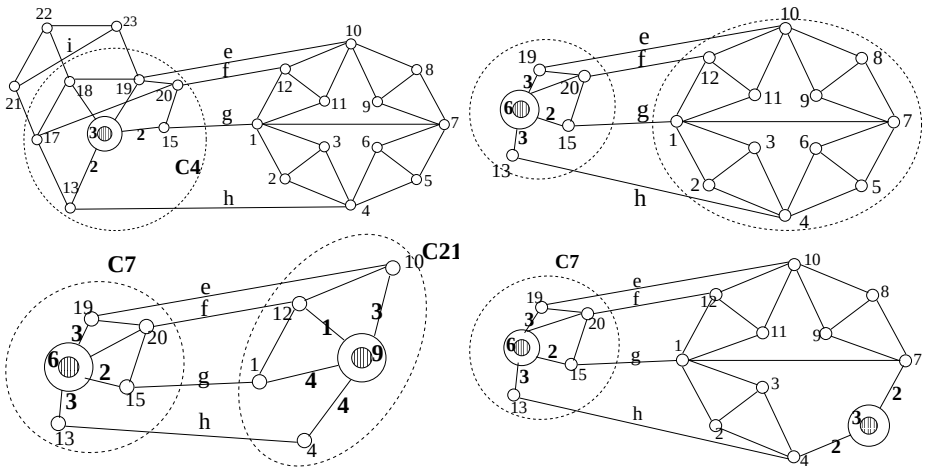

Figure 3–9: Frontier Algorithm's simplification of graph giving DR-plan (Figure 1–5).

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 3–9 illustrates this iterative simplification process ending in the final DR-plan (Figure 1–5).

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.

3.2.3 Datastructures

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

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.

3.2.4 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 3.2.1.

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, that is, its top level clusters are not maximal. The next level of clusters are found by distributing the (completed) clusters currently 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 by Lomonosov and Sitharam [9, 24] that if C is contained inside a

larger cluster, then at least one such cluster will be found by this method once all the clusters currently in the cluster queue have been distributed.

3.2.5 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 1.2. 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 (that is, 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, $N \cup D_1 \cup D_2$, $N \cup D_1 \cup D_2 \cup D_3$ and so on 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 ....

3.3 Incorporating an Input Feature Decomposition into FA DR-plan

We develop a new algorithm that solves the algorithmic problem of Section 3.1. That is, it permits the FA DR-planner (sketched in Section 3.2 [9, 24]) to incorporate into its output DR-plan an input feature decomposition or conceptual, design decomposition.

3.3.1 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.

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." that is, 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 (Figure 3–10), 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.

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 (Section 3.2) and the output DR-plan efficiently maintained and modified as each G_i is worked on, so that the above requirements are satisfied.

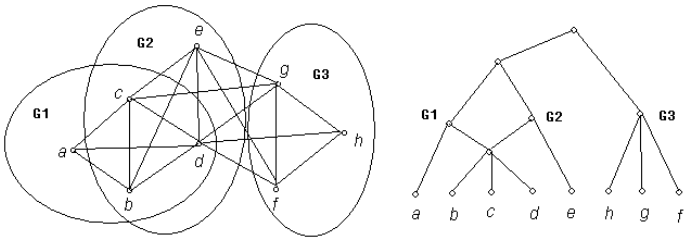

Figure 3–10: Consistency between the DR-plans of 2 groups of graph (Figure 3–6).

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.

3.3.2 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

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.

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 3.2; 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 are provided by Sitharam [11].

The old FA DR-planner of Section 3.2 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; that is 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).

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.

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 3.1), since a proper subset of the children would already form the cluster.

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 3.2), 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.

Once the DR-plans of all the G_i's have been combined, the DR-planner proceeds as described in Section 3.2 on the resulting flow or cluster graph of G, performing *DistributeCluster* on the clusters in them, potentially isolating and

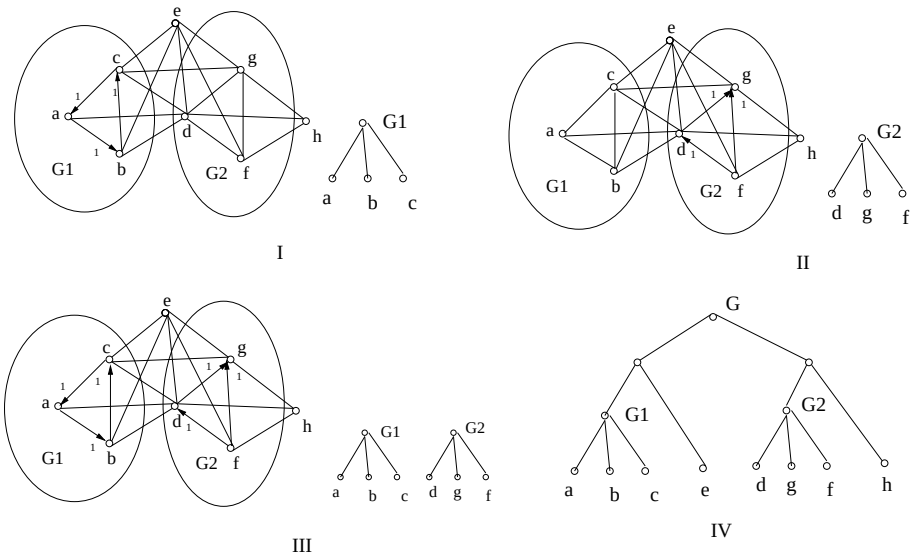

Figure 3–11: Child Clusters do not overlap (Case 1).

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. For the 3D example of points and distance constraints in *Case 1* Part I shows (Figure 3–11) the flow graph and the cluster queue (Section 3.2) 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* (Figure 3–12), 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

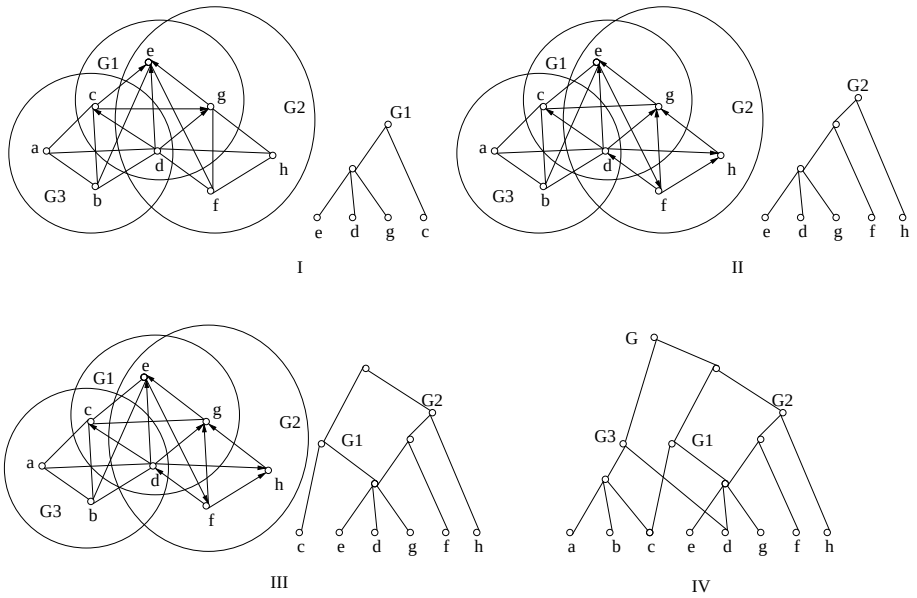

Figure 3–12: Input groups overlap on frontier vertices (Case 2).

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.

For the 3D example of points and distance constraints in *Case 3* Part I shows (Figure 3–13) 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.

3.3.3 Proof of Correctness and Complexity

We show how the 3 requirements of Section 3.3.1 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 3.2, that each feature G_i appears in the output DR-plan provided

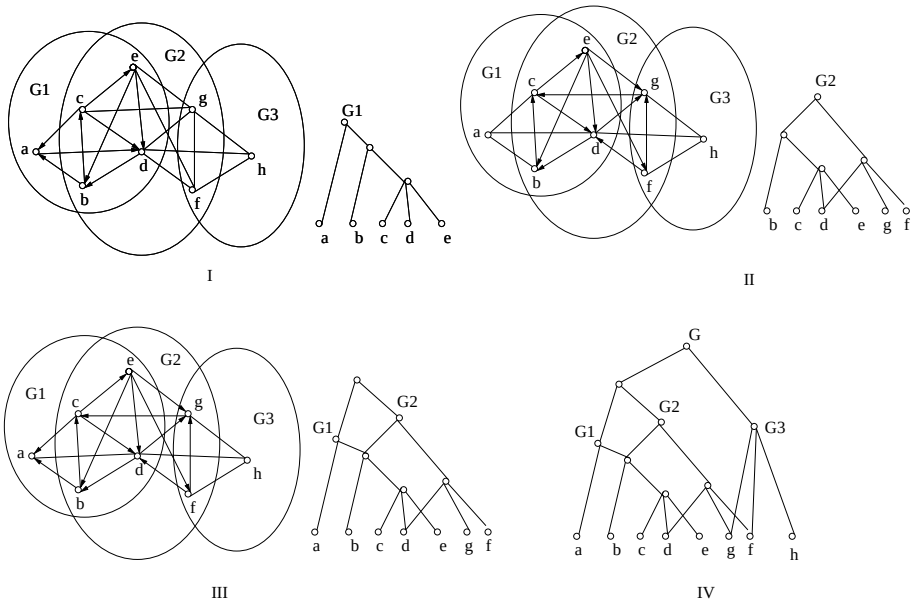

Figure 3–13: The overlapped part includes non-frontier vertices (Case 3).

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.

3.3.4 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 1.2, simply because the old DR-planner is called by the *DistributeGroups* driver at each stage to actually construct the DR-plan. These include properties such as detection and characterization of over and under constrainedness [6, 8], completeness [9, 24], systematic correction of underconstrainedness by giving so-called *completion constraints* amenability to efficient updates (addition or deletion or modification) of geometric primitives or constraints [2, 11], or navigation of the solution space [13].

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.

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 and so on.) 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.

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 [6]. *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 at most a constant factor larger than the optimum.

Also, as mentioned earlier, the feature incorporation does not affect he completeness property, so the algorithms [14, 48] can still be used to find an optimal covering set.

Another property of FA DR-plans that is superficially affected by the presence of features is the systematic correction of overconstraints, that is, the method presented [8]. 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, that is, the set of so-called *reducible* overconstraints is smaller. However, the overconstraint correction method [8] explicitly provides a list of reducible overconstraints *directly associated* with each cluster in the DR-plan. 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.

Finally, a property listed under the output requirements of the feature incorporation problem in Section 3.1 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 (Section 3.2). 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 by Sitharam [2, 11]. 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. that is, 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.

A detailed pseudocode that includes the new feature incorporation algorithm is provided by Sitharam [11]. Documented opensource code can be downloaded from Sitharam's website [15] (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 that the dof-rigid FA DR planner (Section 1.2) considered here does not deal with implicit constraint dependences. However, the more general, module-rigid FA DR-planner [12] 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 [12] has been implemented in FRONTIER [10, 15], 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.

CHAPTER 4
COMBINATORIAL 3D RIGIDITY CHARACTERIZATION

One big open problem is the question of rigidity of 3D constraint systems even when only distances are involved. The problem is that most combinatorial characterizations of rigidity will miss hidden dependencies between constraints and will be wrongly classify many classical nonrigid systems as rigid.

In this chapter, we give a polynomial time characterization called *module-rigidity* that is not fooled by any known such counterexample of nonrigid systems. 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" [19] 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.

4.1 Determining Dof Rigidity: The Frontier Vertex Algorithm (FA)

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) [6, 9] which follows this characterization.

Let C be a geometric constraint graph. Then $Q = \{C_1, \ldots, C_m\}$, 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 4.1.1 *Let C be a geometric constraint graph and $Q = \{C_1, \ldots, C_m\}$, 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.

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

Lemma 4.1.2 *If C is not dof rigid, then only Case 2 in the Definition 4.1 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 Theorem 4.1.1 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 6.1, 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

4.1.1 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: *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.

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

The isolation algorithm, first given by Hoffmann et al. [1, 5] is a modified incremental network maximum flow algorithm. The key routine is the *distribution* of an edge 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." 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 3–8, 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, Hoffmann et al. [1, 5] give an efficient algorithm to find a minimal (non-trivial, if one exists) dof cluster inside it. Then Lomonosov and Sitharam [9] gives a method to ensure a complete, maximal, dof rigid decomposition of C.

The simplification was given by Hoffmann et al. [6, 7]. The found cluster C interacts with the rest of the constraint graph through its *frontier vertices*; that is, 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 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 3–9 illustrates this iterative simplification process ending in the final DR-plan (Figure 1–5).

4.1.2 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 by Hoffmann et al. [6, 7] 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-rigidity algorithm of this section) is given by Sitharam [11, 15]. The pseudocode has been implemented as part of the downloadable, opensource FRONTIER geometric constraint solver [2, 10, 11, 15].

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: outputting 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 (below gives 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* 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. 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, that is, its top level clusters are not maximal. The next level of clusters are found by distributing the clusters currently 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 at least 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. Lomonosov and Sitharam give the formal proofs [9].

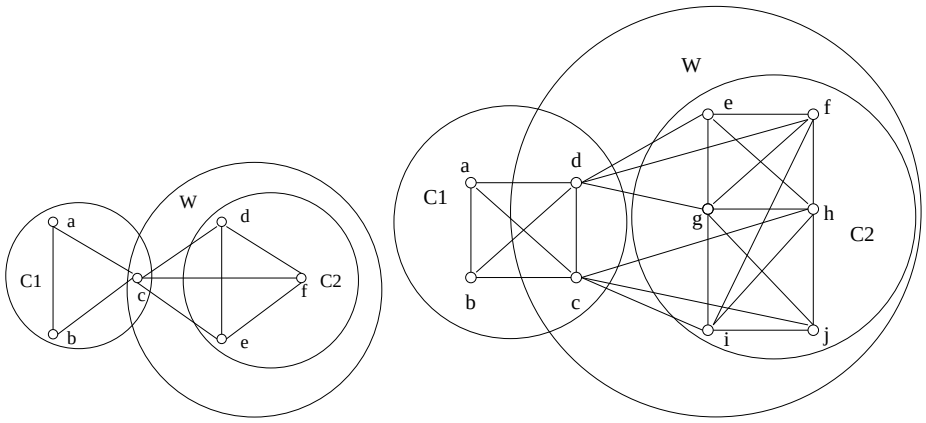

Figure 4–1: Finding W first will prevent dof misclassification with 2D and 3D example.

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 at least 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.

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

First we intuitively explain why dof completeness is a crucial property. After C_1 and C_2 are found (Figure 4–1), 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 by Lomonosov and Sitharam [9] that this type of misclassification can be avoided (W can be forced to be found after C_2 is found), by maintaining

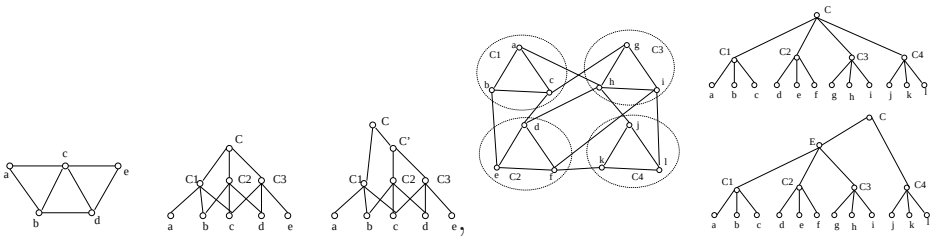

Figure 4–2: Ensuring Cluster Minimality.

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 are given by Lomonosov and Sitharam [9]).

The second invariant that is useful for ensuring dof completeness is that for any cluster in the DR-plan, no proper subset of at least 2 of its child clusters forms a cluster. We call this property *cluster minimality*. FA ensures this using a generalization of the method *Minimal* given by Hoffmann et al. [1, 5] which finds a minimal dense subgraph inside a dense subgraph located by *DistributeVertex* and *DistributeEdge*.

Once a cluster C is located and has children $C_1, \ldots, C_k$, 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 at least 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*. That is, 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.

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. Lomonosov and Sitharam give the proofs [9].

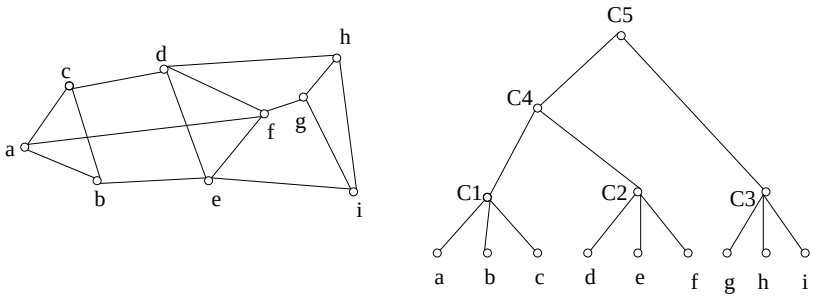

Figure 4–3: Prevent accumulation of clusters

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 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.

When the DR-plan finds (Figure 4–3)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, $N \cup D_1 \cup D_2$, $N \cup D_1 \cup D_2 \cup D_3$ and so on 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 ....

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

4.2 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 $\Rightarrow^{m,C}$, that is, if there is a proper subgraph E of C such that either it is module-rigid, or there is some set of module-rigid proper subgraphs $C_1, \ldots, C_k$ of C such that $C_1, \ldots, C_k \Rightarrow^{m,C,*} E$, then E belongs to $\phi^{m,C,*}$. Let Q be any subset $\{C_1, \ldots, C_m\}$ of $\phi^{m,C,*}$ such that *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 4.2.1 *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 4.2.2 *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 4.2 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. That is, 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. That is, the order of bottom-up construction of module(-complete) DR-plans is immaterial, a type of Church-Rosser property.

Theorem 4.2.3 *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.*

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 4.1. 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. In Figure 4–4 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

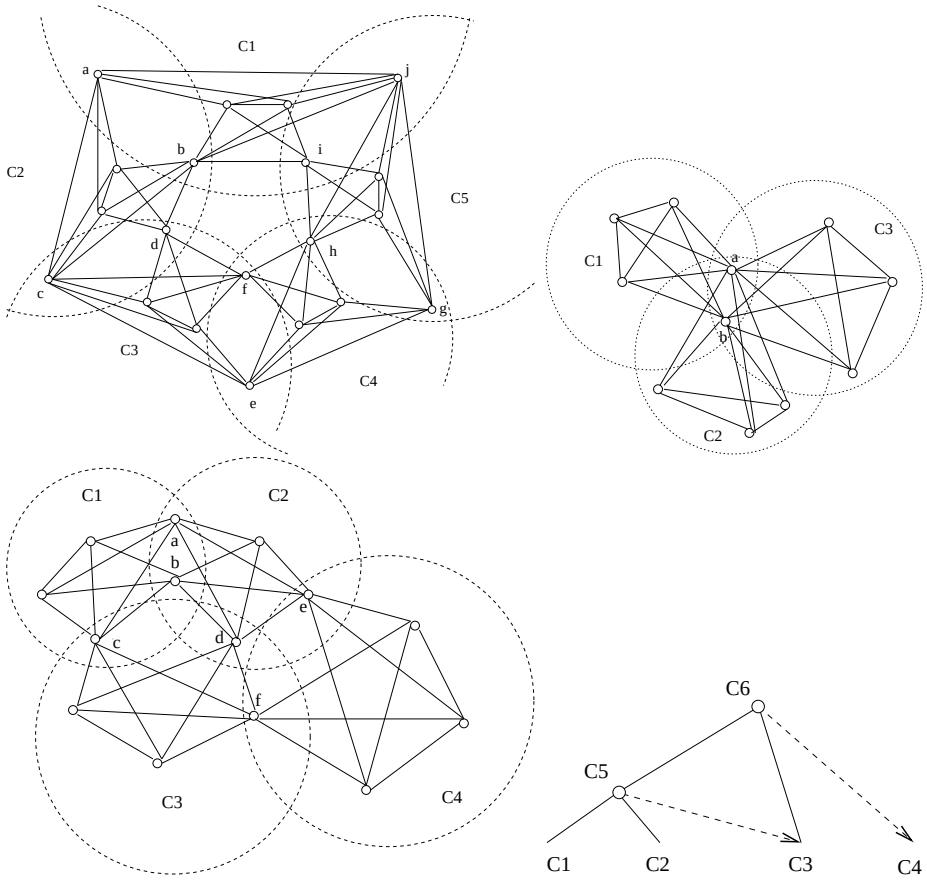

Figure 4–4: Examples where module-rigidity beats dof rigidity.

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; That is, 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 4–5 shows a classic graph given by Crapo et al. [20, 21], 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.

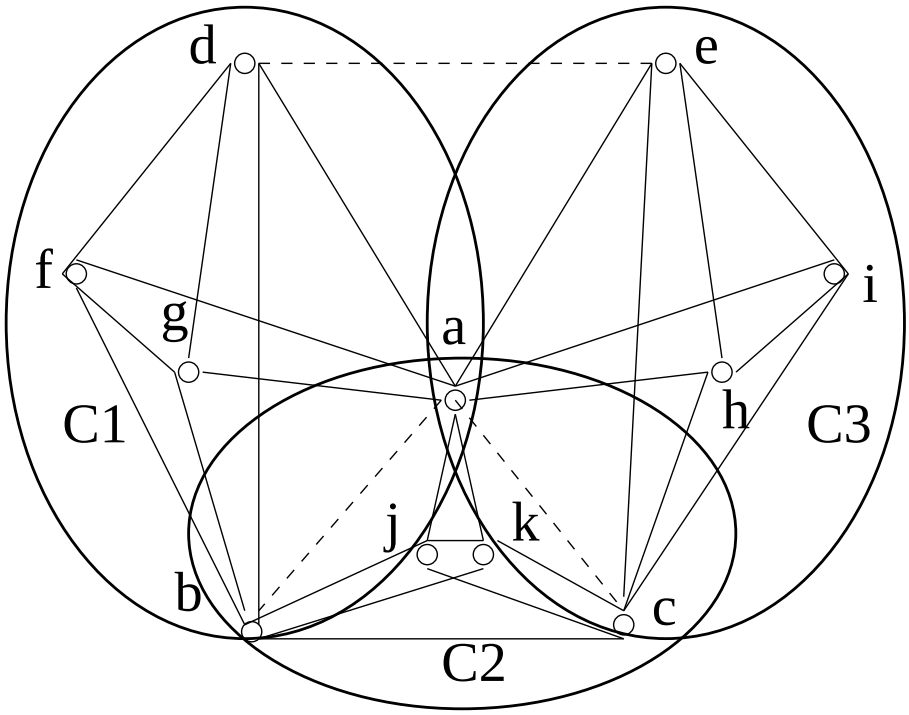

Figure 4–5: Classic Hinge example: not module-rigid, but dof rigid

.

A question that immediately arises is to relate the characterization given here to rigidity matroids and standard conjectures on combinatorial rigidity characterizations for 3D [19].

CHAPTER 5
CONJECTURE OF ANGLE CONSTRAINT SYSTEM

Laman shows a combinatorial characterization of 2D distance constraint system [18]. When other constraints are involved, *for example, angles*, there is no combinatorial characterization. In this chapter, we first give a combinatorial rigidity characterization of a class of angle constraint system. We then show this characterization is not sufficient for the general angle constraint system. Thereafter, we propose a conjecture to determine the rigidity of the general angle system.

5.1 Definitions

Point: point $p \in R^2$, is represented as (x, y).

Angle Constraint: the angle constraint between 2 ordered pairs of points, (P_{1i}, P_{2i}) and (P_{1j}, P_{2j}), satisfies that

$$\cos\theta = \frac{(P_{1i},P_{2i})\bullet(P_{1j},P_{2j})}{|(P_{1i},P_{2i})||(P_{1j},P_{2j})|} = \frac{(x_{2i}-x_{1i})(x_{2j}-x_{1j})+(y_{2i}-y_{1i})(y_{2j}-y_{1j})}{\sqrt{(x_{2i}-x_{1i})^2+(y_{2i}-y_{1i})^2}\sqrt{(x_{2j}-x_{1j})^2+(y_{2j}-y_{1j})^2}}.$$

Geometric Angle Constraint System: a geometric system in which the objects are finite points and the constraints are finite angle constraints.

Solution: the set values of the variables x_i, y_i of the system that satisfy the constraints

Valid Solution: the solution of a geometric system which does not make any angle 0 or points coincident is called a valid solution.

Angle Graph: For a geometric angle constraint system, the graph in which the vertices represent the pairs of points and the edges represent the angles is called the angle graph (Figure 5–1).

Angle Cycle: For a geometric constraint system, if there is an angle cycle in its corresponding angle graph, we say this angle constraint system has an angle

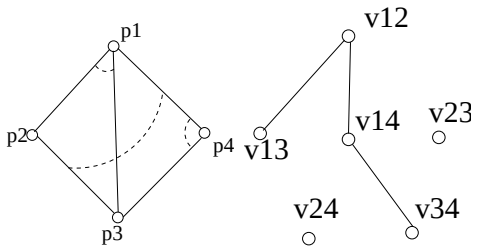

Figure 5–1: Geometric angle constraint system and the angle graph.

cycle. If the angle graph has one more edge between v_{13} and v_{14} (Figure 5–1), its corresponding angle constraint system has an angle cycle.

Angle Pseudo Cycle: For a geometric constraint system, if there is an angle cycle with one angle missing, we say this angle constraint system has an angle pseudo cycle. The system (Figure 5–1) has an angle pseudo cycle.

Angle Clique: For a geometric constraint system, if there exists n points ($n \geq 3$) in which any two pairs of points has an angle constraint between them, we say the system has an angle clique.

Implicit Angle Cycle: In an angle constraint system, for an angle cycle consisting of assigned angles $\theta_1 \ldots \theta_s$ and unassigned angles $\gamma_1 \ldots \gamma_t$, $s > 0, t \geq 0$, in its corresponding angle graph, if there exists $\theta_i \notin S_{\gamma_j}$ for $i \neq j$, where S_{γ_j} is the minimal 4 dof subgraph containing γ_j, we say this angle constraint system has an implicit angle cycle. When $t = 0$, implicit angle cycle turns out to be an angle cycle.

Degrees of Freedom of Primitive Geometric Objects: degrees of freedom (DOF) describes flexibility of motion. It is the number of parameters of the object which may be independently varied. In $2D$, a point has 2 DOF.

Degrees of Freedom of a Geometric System: For a geometric system G, its DOF equals $\sum_{p \in G}$ DOF of object $p - \sum_{c \in G}$ DOF removed by constraint c. In Figure 5–1, the system has 4 points and 3 angle constraints. Each point has 2 DOF and each angle constraint removes 1 DOF. So the system has 5 DOF.

Adjusted Degrees of Freedom (adof): For a geometric system, count the dof of the subgraphs that are angle cliques as 4, then do the inclusion-exclustion dof counting, the total dof is called adjusted degrees of freedom.

Generically independent: Let S be the set of angles of a geometric angle constraint system. The system G^s is generically independent *iff* there is an assignment $S^* = (\theta_1^*, \ldots, \theta_s^*)$ of values to the set S such that G^s has a valid solution and for any such assignment S^*, there is a rectangle neighborhood of S^*, that is, $I_S = I_{\theta_1} \times \ldots \times I_{\theta_s}$, where the $I_{\theta_l} = (\theta_l^* - \epsilon, \theta_l^* + \epsilon)$ are intervals of angle values, for some $\epsilon > 0$, such that for any tuple of values $(\theta_1', \ldots, \theta_s') \in I_S$, G^s has a valid solution.

Notations: We use G_n^s to denote a geometric angle constraint system with n points; G_n to denote the corresponding angle graph of G_n^s; p to denote the point in G_n^s; v to denote the vertex in G_n.

5.2 One Combinatorial Characterization

In this section, we give a combinatorial characterization of the angle constraint system that can be obtained by the following construction.

Gradual Construction: Starting with a pair of points, add a point with at most 2 angles connecting it with the constructed system at each step until all points are added.

If an angle constraint system G^s can be constructed by gradual construction, it is *gradually constructible*.

Theorem 5.2.1 *An gradually constructible geometric angle constraint system is generically independent if it does not have implicit cycles.*

Proof

We prove it by induction on number of points n of G^s:

1. Base case: when G^s has only 2 points, it is obviously generically independent.

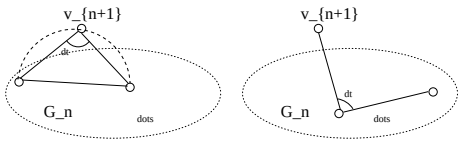

Figure 5–2: The new point p_{n+1} is at the circle or at the radial

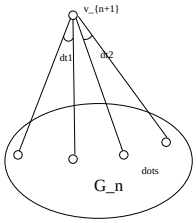

Figure 5–3: Two new angles: δ_1 and δ_2

2. Assume G_n^s has T angles and is generically independent, because G^s is gradually constructible, we can find a point p_{n+1} with at most 2 angles connecting it with G_n^s.

Case 1: there is no angle connecting p_{n+1} with G_n^s.

Obviously, the case holds.

Case 2: there is an angle connecting p_{n+1} with G_n^s.

p_{n+1} could be at the circle or at the radial determined by the angle δ (Figure 5–2). It is easy to know there exists a value of δ which together with the T angles which make G_n^s have a valid solution that make G_{n+1}^s have a valid solution. From hypothesis we know, when G_n^s has a valid solution, these is a corresponding rectangle I_T such that G_n^s with any tuple of values $(\theta_1', \ldots, \theta_t') \in I_T$ has a valid solution. As the position of p_{n+1} is continuous, there exists an interval I_δ for δ, such that G_{n+1}^s with any tuple of values in $I_T \times I_\delta$ has a valid solution. So, G_{n+1}^s is generically independent.

Case 3: there are 2 angles connecting p_{n+1} with G_n^s.

Call the 2 angles δ_1 and δ_2. We prove case 3 in the following 3 subcases. We call the angles involved in p_{n+1} *new* angles and the others *old* angles.

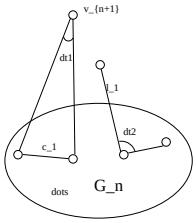

Figure 5–4: One new angle δ_1 and one old angle δ_2

Subcase 1: Both δ_1 and δ_2 are new angles (Figure 5–3). The position of p_{n+1} should be at the intersection of two circles which are determined by δ_1 and δ_2 respectively. For a $T \in I_T$, choose $\delta_i \in (0, \epsilon^T_{\delta_i})$, $\epsilon^T_{\delta_i} \to 0$ to ensure two circles intersect. If for δ^*_i, p_{n+1} is in a corresponding line of a pair of points of G^s_n, we choose $\epsilon^T_{\delta_i}$ equal this δ^*_i to prevent p_{n+1} lie in this line. After proceeding with all the lines, we ensure p_{n+1} does not lie in any line of G^s_n. If δ^*_i, p_{n+1} coincidents with a point of G^s_n, we choose $\epsilon^T_{\delta_i}$ equal this δ^*_i to prevent p_{n+1} coincident with this point. After proceeding with all the points, we ensure p_{n+1} does not coincident with any point in G^s_n. Let $I_{\delta_i} = \bigcap_{T \in I_T}(0, \epsilon^T_{\delta_i})$ and $I_\Delta = I_{\delta_1} \times I_{2\delta_2}$. We know G^s_{n+1} with any tuple of values in $I_T \times I_\Delta$ has a valid solution. So G^s_{n+1} is generically independent.

Subcase 2: δ_1 is a new angle and δ_2 is an old angle (Figure 5–4). The position of p_{n+1} should be at the intersection of the circle which is determined by δ_1 and the radial which is determined by δ_2. For a $T \in I_T$, choose $\delta_1 \in (0, \epsilon^T_{\delta_1})$, $\epsilon^T_{\delta_1} \to 0$. $\delta_2 \in (\epsilon^T_{\delta_2}, \pi)$. $\epsilon^T_{\delta_2} \to \pi$ to ensure the radial intersects the circle. If for δ^*_i, p_{n+1} is in a corresponding line of a pair of points of G^s_n, we choose $\epsilon^T_{\delta_i}$ equal this δ^*_i to prevent p_{n+1} lie in this line. After proceeding with all the lines, we ensure p_{n+1} does not lie in any line of G^s_n. If δ^*_i, p_{n+1} coincidents with a point of G^s_n, we choose $\epsilon^T_{\delta_i}$ equal this δ^*_i to prevent p_{n+1} coincident with this point. After proceeding with all the points, we ensure p_{n+1} does not coincident with any point in G^s_n. Let $I_{\delta_1} = \bigcap_{T \in I_T}(0, \epsilon^T_{\delta_1})$ and $I_{\delta_2} = \bigcap_{T \in I_T}(\epsilon^T_{\delta_2}, \pi)$ and $I_\Delta = I_{\delta_1} \times I_{2\delta_2}$. We know G^s_{n+1}

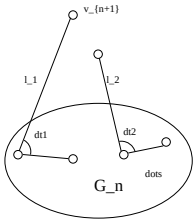

Figure 5–5: Two old angles do not share a pair of points containing p_{n+1}

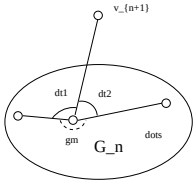

Figure 5–6: Two old angles share a pair of points containing p_{n+1}

with any tuple of values in $I_T \times I_\Delta$ has a valid solution. So G^s_{n+1} is generically independent.

Subcase 3: Both δ_1 and δ_2 are old angles (Figure 5–5). The position of p_{n+1} should be at the intersection of two radials which are determined by δ_1 and δ_2 respectively. For a $T \in I_T$, choose $\delta_1 \in (0, \epsilon^T_{\delta_1})$, $\epsilon^T_{\delta_1} \to 0$ and choose $\delta_2 \in (\epsilon^T_{\delta_2}, \pi)$, $\epsilon^T_{\delta_2}$ makes $l_1 \parallel l_2$ to ensure two radials intersect. If for δ^*_i, p_{n+1} is in a corresponding line of a pair of points of G^s_n, we choose $\epsilon^T_{\delta_i}$ equal this δ^*_i to prevent p_{n+1} lie in this line. After proceeding with all the lines, we ensure p_{n+1} does not lie in any line of G^s_n. If for δ^*_i, p_{n+1} coincidents with a point of G^s_n, we choose $\epsilon^T_{\delta_i}$ equal this δ^*_i to prevent p_{n+1} coincident with this point. After proceeding with all the points, we ensure p_{n+1} does not coincident with any point in G^s_n. Let $I_{\delta_1} = \bigcap_{T \in I_T}(0, \epsilon^T_{\delta_1})$ and $I_{\delta_2} = \bigcap_{T \in I_T}(\epsilon^T_{\delta_2}, \pi)$ and $I_\Delta = I_{\delta_1} \times I_{2\delta_2}$. We know G^s_{n+1} with any tuple of values in $I_T \times I_\Delta$ has a valid solution. So G^s_{n+1} is generically independent.

Note that δ_1 and δ_2 can not be 2 old angles sharing a pair of points containing p_{n+1} (Figure 5–6), because G has no implicit angle cycle.

From above, we know G^s_{n+1} is generically independent in case 3.

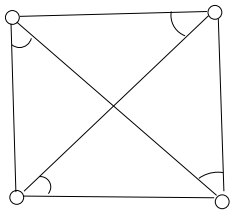

Figure 5–7: Generically independent system that is not gradually constructible

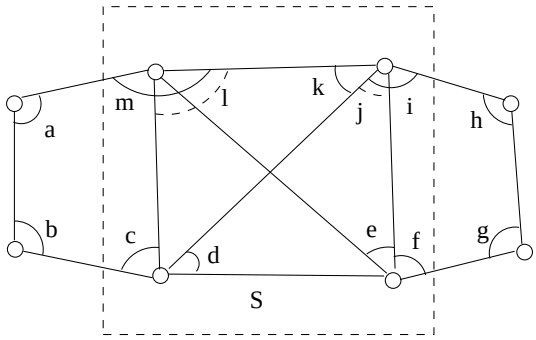

Figure 5–8: Implicit angle cycle is not the only type of dependency.

So if G is gradually constructible and has no implicit angle cycle, it is generically independent.

On the other hand, if G is not gradually constructible, it may still be generically independent. G (Figure 5–7) is not gradually constructible because each point has 3 angles involved. But G is still generically independent.

The implicit angle cycle is not the only type of dependency in the angle constraint system. Angle l (Figure 5–8) is fixed by angles a, b, c, m and angle j is fixed by angles f, g, h, i. Then the subgraph S actually has 3 degrees of freedom because it has 5 angles (d, e, j, k, l) fixed. Thus S is not generically independent.

Because detecting the implicit angle cycle is not sufficient to tell whether an angle constraint system is generically independent, we propose a rigidity conjecture.

5.3 Observations

The following two observations are crucial for the proof of the conjecture. First, we give an algorithm that is essential in our discussion.

Verifier: Taken a geometric angle constraint system G as input,

1. If G has a 3 or fewer dof subgraph, return *false*
2. If there exists an angle cycle which is not contained in an angle clique, return *false*
3. Find an angle pseudo cycle if it exists. Assign the missing angle.
4. If there exists one subgraph with 3 or fewer adof, return *false*.
5. Find a 4 adof subgraph that is not an angle clique if it exists. Assign all the unassigned angles .
6. repeat 2, 3, 4 and 5, until no more angles are assigned, return *true*

Note that Verfier may have diffirent computational paths based on which angle cycle chosen in Step 3 and which 4 adof sugraph chosen in step5.

Observation 5.3.1 *For an unassigned angle γ that is assigned in a path P_a of Verifier, it will be assigned in any path of Verifier unless Verifier returns false before it encounting γ.*

Assume there exists a path P_b in which γ is not assigned and Verifier returns true. P_b does not assign all the angles assigned in P_a otherwise γ is also assigned in P_b violating the assumption. Let θ be the first angle assigned in P_a but not assigned in P_b. All the angles assigned before θ in P_a that are necessary for assigning θ are also assigned in P_b. Verifier only returns true if there is no more angle to be assigned. But at the end of P_b, Verifier can still assign θ because all angles necessary for assigning θ have been assigned in P_b. That contradicts the assumption that Verifier returns true at the end of P_b.

Observation 5.3.2 *All paths return the same result.*

Assume there are two paths P_a and P_b that P_a returns false and P_b returns true. P_b does not assign all the angles assigned in P_a otherwise P_b will also return false. Let θ be the first angle assigned in P_a but not assigned in P_b. All the angles assigned before θ in P_a that are necessary for assigning θ are also assigned in P_b. Verifier only returns true if there is no more angle to be assigned. But at the end of P_b, Verifier can still assign θ because all angles necessary for assigning θ have been

assigned in P_b. That contradicts the assumption that Verifier returns true at the end of P_b.

Observation 5.3.3 *All paths that return true assign the same set of angles.*
Assume there are two paths P_a and P_b that both return true but they assign diffrent sets of angles. Let θ be the first angle assigned in P_a but not assigned in P_b. All the angles assigned before θ in P_a that are necessary for assigning θ are also assigned in P_b. Verifier only returns true if there is no more angle to be assigned. But at the end of P_b, Verifier can still assign θ because all angles necessary for assigning θ have been assigned in P_b. That contradicts the assumption that Verifier returns true at the end of P_b.

Observation 5.3.4 *Verifier has Church-Rosser property.*
From Observation 5.3.2, we know Verifier has Church-Rosser property.

Observation 5.3.5 *Verifier has polynomial time complexity.*
To use MFA algorithm find 3 or fewer adof subgraph in step 1,4 and the 4-adof subgraph in step 5, it requires $O(n^3(n+m))$ (n is the number of vertices, m is the number of edges). To use depth-first search to find angles in step 2, 3, it requires $O(n+m)$. Verifier runs at most n^4 iterations because there are at most n^4 angles and at least one angle is assigned at each iteration, then the time complexity is $O(n^7(n+m))$. Actually the upbound of the time complexity is very loose.

If the verifier returns true for a geometric angle constraint system, we say the system passes the verifier. Then we have the following observations.

Observation 5.3.6 *If a geometric angle constraint system G fails the verifier, it is not generically independent.*

If G fails the verifier at step 1, it is obvious that the angles in the 3 or fewer dof subgraphs are not independent to each other, thus G is not generically independent.

If G fails at step 2, then for any angle of the angle cycle, its value is fixed by every setting of values of the other angles in the angle cycle, thus G is not generically independent.

If G fails at step 4, it is obvious that the angles in the subgraph with 3 or fewer adof are not independent, thus G is not generically independent.

Observation 5.3.7 *Given an angle constraint system G_n^s in 2D with $k(\geq 0)$ dof, there is at least one point involved in at most 7 angles.*

Let m be the number of angles involve in the point that has the least number of angles involved. As an angle involves in at most 4 points and G_n^s has k dofs, the following inequality holds: $m * n \leq 4 * (2n - k)$. Then we get $m \leq 8 - \lceil 4 * k/n \rceil$. So $m \leq 7$.

5.4 Conjectures

For the following three conjectures, the first two are used for the proof of the third one which is our 2D rigid characterization conjecture of angle constraints system.

Conjecture 5.4.1 *For an angle constraint system G_n^s with $k(\geq 4 + x)$ dof that passes the verifier, there exists an x-tuple of unspecified angle tuple such that specifying it does not make the system fails the verifier.*

Partial Proof:

Let $G_{S_i}^s$ be the ith maximal 4 dof subsystem in G_n^s which has at least 3 points. Then this property holds: G_{S_i} and G_{S_j} are disjointed for any $i \neq j$, that is $G_{S_i}^s$ and $G_{S_j}^s$ share at most one point.

If G_{S_i} and G_{S_j} share a vertex, $G_{S_i}^s \bigcup G_{S_j}^s$ has 4 dof if no angle involves in both $G_{S_i}^s$ and $G_{S_j}^s$. This violates the statement $G_{S_i}^s$ and $G_{S_j}^s$ are the maximal 4 dof subsystems in G_n^s; Or $G_{S_i}^s \bigcup G_{S_j}^s$ has 3 or fewer dof if at least one angle involves in both $G_{S_i}^s$ and $G_{S_j}^s$. This is impossible because no subsystem has 3 or fewer dof in G_n^s. So the property holds.

G_n has t trees: $t = n(n-1)/2 - (2n-k) = ((n-2.5)^2 - 6.25)/2 + k$. So when k is given, t is increasing for $n \geq 3$. Note that the tree could be consist of only one vertex. When $n = 3$, $t = k - 3 \geq x + 1$, that is for a given G_n with k dof, it has at least $x + 1$ trees.

We give an algorithm to find the x-tuple of unspecified angles without forming an angle cycle or making any subgraph 3 or fewer dofs. Here is its pseudocode.

```
While x>0
   Find maximal 4 dof subsystems
   If there is no $G_{S_i}$ in $G_n$
     add an angle between a pair of trees arbitrarily
   else
     add an angle between (v1, v2), where v1 is in $G_{S_i}$
     and v2 which is in another tree of v1 and outside
     of $G_{S_i}$
   x--
end
```

If there is no $G^s_{S_i}$ in G^s_n at current stage, then we can choose the angle in the x-tuple of unspecified angles between a pair of trees arbitrarily without forming an angle cycle or making any subgraph 3 or fewer dofs. If there is a $G^s_{S_i}$, because the trees can't be totally contained in G_{S_i} all together, we can always find such a pair (v_1, v_2) such that v_1 is in G_{S_i} and v_2 is in another tree of v_1 and outside of G_{S_i}. That ensures it does not form an angle cycle. And because G_{S_i}s are disjointed, it does not make any subsystem 3 or fewer dof.

We still need to prove G^s_n with specifying that x-tuple passes the verifier.

Conjecture 5.4.2 *For an angle constraint system G^s_{n+1} that passes the verifier, if $m \geq 4$, m is the number of angles involved in point p_{n+1} that involves the least angles, then there exists an unspecified angle γ in G^s_n such that after unspecifying one of the m angles involved in p_{n+1} to get G'^s_{n+1} and specifying γ in G'^s_{n+1} to get G''^s_{n+1}, G''^s_{n+1} passes the verifier.*

Partial Proof:

From the proof of Observation 5.3.7, we know $n+1 \geq \lceil 4k/(8-m) \rceil$. When $m = 4$, $n \geq 3$. But when $n = 3$, no G_4^s satisfies that the number of angles involved in the point that involves the least angles is 4. In fact, it can be at most 3. So $n \geq 4$. When $m = 4 \ldots 7$, $n > 4$. Then we know G_n^s has at least 4 points.

G_n^s has $k_n = k + (m-2)$ dofs. And G_n has $t = n * (n-1)/2 - (2n - k_n) = (n^2 - 5n)/2 + k_n$ trees. When $n = 4$, $t = m + (k-4) \geq m$. Then the $m-1$ edges of $G'_{n+1} \setminus G_n$ is not enough to connect all these t trees to be a tree of G'_{n+1}. Then we know the vertices in G_n can't be all in a tree of G'_{n+1}.

Let $G_{S_i}^s$ be the ith maximal 4 dof subsystem in G_{n+1}^s which has at least 3 points. If p_{n+1} is not in any $G_{S_i}^s$, then we add γ by the way similar to Conjecture 5.4.1. If p_{n+1} is in a $G_{S_i}^s$, we add γ between the (l_1, l_2), l_1 is the edge in $G_{S_i}^s$ and l_2 is in $G_n^s \setminus G_{S_i}^s$ and l_1 and l_2 are not in the same tree of $G_{n+1}^{\prime s}$. This pair does exist because the vertices in G_n can't be all in a tree of G'_{n+1}. So $G_{n+1}^{\prime\prime s}$ has no angle cycles and no subsystem has 3 or fewer dof.

Also, we need to prove $G_{n+1}^{\prime\prime s}$ passes the verifier.

Conjecture 5.4.3 *For an angle constraint system G^s in 2D that passes the verifier, it is generically independent.*

Partial Proof:

Case 1: there exists a point p_{n+1} has no angle involved. Case 2: there exists a point p_{n+1} which has 1 angle involved. Case 3: there exists a point p_{n+1} which has 2 angles involved. In these 3 cases, $G_{n+1)}^s$ is generically independent. Refer to the proof of Theorem 5.2.

Case 4: there exists a point p_{n+1} which has $m(\geq 3)$ or more angles involved.

ASSUMPTION: For an angle constraint system G^s in 2D with 4 dof and no angle cycles and no subsystem has 3 or fewer dof, and G^s with $I_a = I_1 \times I_2 \times \ldots \times I_s$ has a valid solution, I_i is the interval of the specified angle i in G^s, s is the number of specified angles, for an unspecified angle δ specifying which does not make any

angle cycle, let γ be the specified angle which is in the minimal 4 dof subsystem that contains all the lines participating δ, then for any $v^* \in I_{a\setminus\gamma}$, as γ varies in I_γ, there exists a non-zero sized I_δ for δ.

If $m = 3$, call the three angles δ_1, δ_2 and δ_3. From Conjecture 5.4.1, we know that there exists an unspecified angle γ in G_n^s that does not form an angle cycle involving some subset of the specified angles or make any subsystem 3 or fewer dof. Unspecify δ_3 and specify γ to get $G_{n+1}'^s$. From case 3, we know $G_{n+1}'^s$ with $I_a = I_1 \times I_2 \times \ldots I_\gamma \times \ldots \times I_s \times I_{\delta_1} \times I_{\delta_2}$ has a valid solution, s is the number of angles in $G_n'^s$. And the minimal 4 dof subsystem of δ_3 must contain γ. Then from the assumption, we know for any $v^* \in I_{a\setminus\gamma}$, as γ varies in I_γ, there exists a non-zero sized $I_{\delta_3}^{v^*}$ for δ_3.

Let $I_i' \in I_i$ and $|I_i'| \to 0$, i is the specified angle in $G_{n+1}'^s$. As v^* varies in $I'_{a\setminus\gamma}$, $I_{\delta_3}^{v^*}$ does not change much because $I'_{a\setminus\gamma}$ is relatively small. Then $I_{\delta_3} = \bigcap_{v^* \in I'_{a\setminus\gamma}} I_{\delta_3}^{v^*}$ has non-zero size.

Then we know G_{n+1}^s with $I'_{a\setminus\gamma} \times I_{\delta_3}$ has a valid solution, so G_{n+1}^s is generically independent.

If $m = 4$, call the 4 angles $\delta_1, \delta_2, \delta_3$ and δ_4. Choose γ as described in Conjecture 5.4.2.

After specify γ and unspecify δ_4 to get $G_{n+1}'^s$, and choose I_i' similar to the case when $m = 3$, we know I_{δ_4} that has non-zero size exists.

Then G_{n+1}^s with $I'_{a\setminus\gamma} \times I_{\delta_4}$ has a valid solution, so G_{n+1}^s is generically independent.

We can also do the similar job for $m = 5 \ldots 7$.

CHAPTER 6
SOLUTION SPACE NAVIGATION

In this chapter, we study the well-documented problem of systematically navigating the potentially exponentially many roots or realizations of well-constrained, variational geometric constraint systems. We give a scalable method called the ESM or Equation and Solution Manager that can be used both for automatic searches and visual, user-driven searches for desired realizations. The method incrementally assembles the desired solution of the entire system and avoids combinatorial explosion, by offering the user a visual walkthrough of the solutions to recursively constructed subsystems and by permitting the user to make gradual, adaptive solution choices.

We isolate requirements on companion methods that permit (a) incorporation of many existing approaches to solution space steering or navigation into the ESM; and (b) integration of the ESM into a standard geometric constraint solver architecture. We address the latter challenge and explain how the integration is achieved. And we clarify these requirements essential and desirable for efficient, meaningful solution space navigation. Also, we sketch the ESM implementation as part of an opensource, 2D and 3D geometric constraint solver FRONTIER developed by our group.

6.1 Companion Methods Requirements

In order to describe the ESM method and how it can be integrated into *any* geometric constraint solver (Figure 6–1), it is necessary to give well-defined *essential* and *desirable* requirements on the companion methods - mentioned in the Introduction - for the ESM method to be effective. We additionally point to known literature that provide the companion methods that meet these requirements.

6.1.1 DR-Planner

In order to give the DR-planner requirements, we first need some preliminaries about combinatorial analysis of geometric constraint systems. 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 (dofs)* available to an object represented by v and number of degrees of freedom removed by a constraint represented by e respectively.

For example, Figure 1–1 shows a 2D constraint systems and their respective dof constraint graphs. Figure 1–4 shows a 3D constraint systems whose graph have vertices of weight 3 (points) and edges of weight 1.

Note that the constraint graph could be a *hypergraph*, each hyperedge representing a constraint involving any number of vertices. A vertex induced subgraph represents a subsystem of the entire constraint system. The subgraphs can be classified as being *underconstrained, well-constrained, or well-overconstrained* - the latter two are called *rigid* subgraphs or *clusters*. The meaning is that the corresponding subsystems have the corresponding constrainedness properties for *generic* parameter values. In many cases, this classification of the subgraphs can be done purely combinatorially, using the dof weights, but ignoring geometric parameters such as distance, angle and so on attached to the constraints.

As mentioned in the introduction, any effective constraint solver combinatorially develops a plan for recursively decomposing the constraint system into small subsystems, whose solutions - obtained from the algebraic/numeric solver - can be recursively recombined by solving other small subsystems. Such a recombination is straightforward, *provided all the subsystems generically have a finite number of solutions, that is, they are generically rigid.*

The DR-planner is typically 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 well-constrained subgraph or cluster 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}. While we rely on a rough correspondence between well-constrained subgraphs or clusters and *rigid* subsystems, the exact nature and limitations of this correspondence is a complex issue, especially in 3D, and is discussed later.

Formally, a DR-plan of a constraint graph G is a directed acyclic graph (DAG) whose nodes represent rigid clusters 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 clusters of G. For well or well-overconstrained graphs, the DR-plans have a single source, and for underconstrained graphs, the DR-plans have multiple sources. There could be many DR-plans for G (Figures 1–1, 1–4). Generally, for overconstrained clusters, a DR-plan is required to also contain information about which sets of overconstraints can be removed while retaining the rigidity of the cluster. These are called *reducible* overconstraints. Reducible overconstraint *directly associated* with a cluster C refer to those that connect primitive elements occurring in different child clusters of C. Overconstraints that lie within any child cluster C_i are associated with that C_i, not with C.

We describe essential and desirable properties that a good DR-plan(ner) should have for it to be useful in conceptual navigation of the solution space of general, cyclic 2D or 3D systems.

It is *desirable* to be able to apply the ESM method to a large class of 2D and 3D constraint systems. For the method to work for any such class, it is *essential* that the DR-plan's nodes should correspond to rigid clusters or generically rigid

subsystems, and all choices of reducible constraints of overconstrained clusters should be available. Recall that these clusters and their reducible constraints are picked by the DR-planner using just the constraint graph, that is, using purely combinatorial properties of the constraint system.

To illustrate the level of challenge involved in doing this: to date, there is no known, tractable, characterization of generic rigidity of distance constraint (sub)systems for 3 or higher dimensions, based purely on combinatorial properties of the constraint graph [19, 22], although several conjectures exist. Moreover, there are no known combinatorial characterizations of 2D rigidity, when other constraints besides distances are involved. It should be noted that in 2D, Laman's theorem [18], gives such a characterization if all geometric objects are points and all constraints are distances. However, the standard generalization of the Laman property for 3D, also called *dof-rigidity* is inadequate: while all rigid systems are dof-rigid, the converse is not the case in 3D. Standard counterexamples are systems that contain constraint dependences or inexplicit overconstraints hidden in so-called "bananas" or "hinges" [19, 20, 21] (Figure 1–4). In fact, these are the only known types of counterexamples.

The *module-rigid* Frontier vertex algorithm described by Sitharam and Zhou [12] and implemented in FRONTIER [15] is the first known polynomial time DR-planner that is not fooled by any known type of constraint dependence, specifically, the "bananas" or "hinge" type constraint dependences. It gives a DR-plan whose nodes are so-called module-rigid clusters. It is an open question whether there are any nonrigid subgraphs that are module-rigid. No counterexamples are known.

The module-rigid Frontier vertex algorithm is crucially based on the so-called *dof-rigid* Frontier vertex algorithm analyzed by Lomonosov and Sitharam [2, 9, 24]. While the latter does not detect bananas and other constraint dependences, it is

the first algorithm that gives a *complete* DR-plan, that is, a *complete decomposition* of each cluster into *maximal* proper subclusters. A generalized version of this decomposition is the key starting point for *detecting* module-rigidity [12] and is also crucial for obtaining a tractable DR-plan that aids efficient solving (discussed next).

The basic idea for the dof-rigid Frontier vertex DR-planners (without a complete formal analysis and without the fully general completeness property) was presented by Hoffmann et al. [6]. 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 by Hoffmann and Sitharam [8].

The *size* of a cluster in a DR-plan is its fan-in (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 by Lomonosov and Sitharam [9, 24], that the problem of finding the optimal DR-plan of even a 2D distance constraint graph is NP-hard, and approximability results are shown only in special cases. Nonapproximability results are not known.

To get around this difficulty, we use 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 the optimal covering set for C is the size of the cluster C, that is, its fan-in in the DR-plan. For scalability of the ESM method, we restrict ourselves to

constraint graphs for which this size is typically constant independent of the original graph size. The *optimality* here refers not only to the size, but also the algebraic complexity of the active constraint system for solving C, given the solutions of its child clusters. This optimization is the function of the combinatorial cluster-system optimizer (CCO) method (Figure 6–1) described later in this section. Note that in order to choose the optimal covering set of child clusters for a cluster C, the CCO needs as input a generalized complete decomposition of C into maximal proper subclusters, which was already seen to be an essential requirement for the generality of a DR-planner, now seen to be essential for tractability as well.

Another crucial property of a DR-plan is its *width* i.e, *number* of clusters in the DR-plan. It is *essential* that this be small, preferably linear, certainly polynomial, in the size of G: this reflects the complexity of the planning process and also affects the complexity of the solving process that is based on the DR-plan. The module-rigid and dof-rigid Frontier vertex DR-planners analyzed by Sitharam and Yong and Lomonosov [2, 9, 12, 24], and implemented in FRONTIER [15], are the only ones that have quadratic (typically linear) width and output complete maximal decompositions of each cluster, whose optimal covering sets have typically constant size.

For the user to effectively navigate the solution space just by inspecting the solutions to subsystems in the DR-plan, it is *desirable* that these subsystems include those (well or well-overconstrained clusters) that occur in an underlying conceptual decomposition. This is a feature, part or subassembly hierarchy that captures design intent, that is, a partial order, typically also represented as a directed acyclic graph.

This incorporation of such an input decomposition is crucial also in order to allow independent and local manipulation of features, parts, subassemblies or subsystems within their local coordinate systems; for allowing the user to dictate

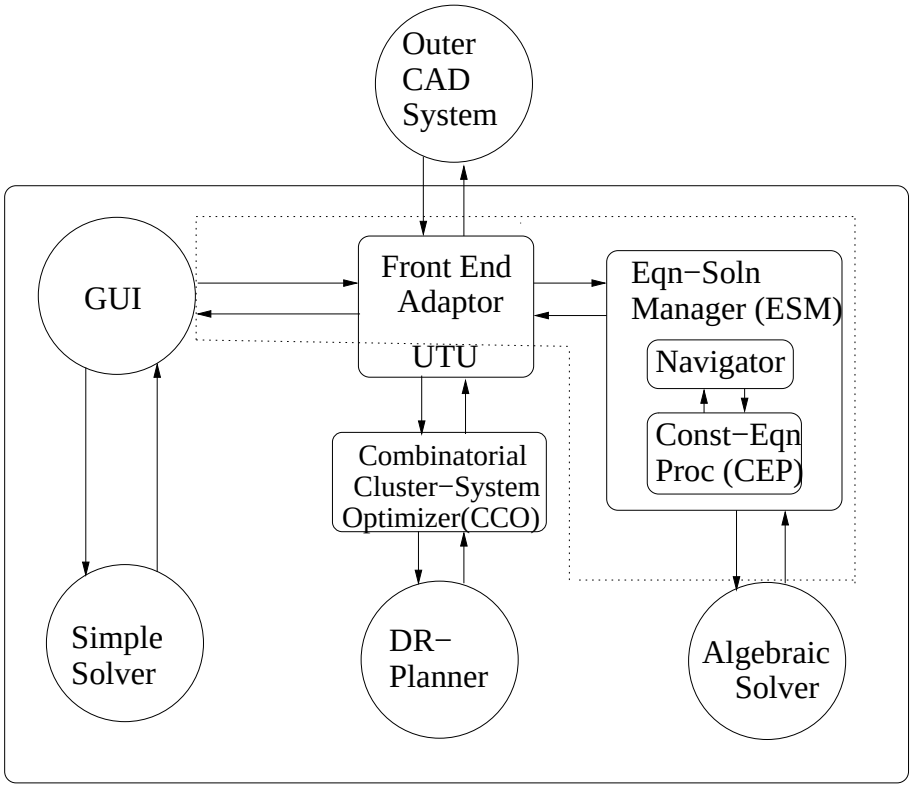

Figure 6–1: Standard geometric constraint solver architecture.

the order of resolution (and solution space inspection) of the features, parts, subassemblies or subsystems. For example, parametric constraint solving can in fact be achieved as a special case, where the order is a complete, total order. In addition such a *solving priority order* arises naturally in the case of module-rigid DR-plans [12] discussed in Section 6.1. These include clusters whose clusterhood depends on first solving other (nondescendant) clusters in the DR-plan.

6.1.2 Combinatorial Cluster-System Optimizer (CCO)

This method is described in detail by Sitharam et al. [14]. Here we give its requirements and output. The CCO method takes its input from the DR-plan (Figure 6–1) one cluster C at a time, along with a complete maximal decomposition into subclusters C_i, and complete information on reducible overconstraints directly associated with each C_i and with C. The CCO method's output consists of: (i) a covering set of clusters among the C_i's that will be used for solving C, (ii) a

subset of the *cluster overlap* constraints for the chosen covering set which together with a well-constrained set of original constraints, result in a system of algebraic equations (a) that is stable or independent and (b) whose complexity - that is, number of variables and degree - has been *combinatorially* minimized; typically this complexity is constant, independent of the size of the constraint graph. Note that the actual algebraic system for solving C is never generated or manipulated by the CCO method. All the output requirements are *essential* for tractability of solution and thus for the ESM method presented here, and are proven by Sitharam et al. [14], except for the Requirement (ii)(a) which is proven by Sitharam [48].

6.1.3 Algebraic Solver

This could be a purely algebraic or numeric solver. An *essential* property of the Algebraic Solver (Figure 6–1) is that it should be reasonably efficient in practice; numerically stable; output all real solutions to the input polynomial system; uses interval arithmetic, that is, can deal with interval values for the coefficients of the polynomials; and can search for solutions within specified intervals for each variable. *Desirable* properties depend on types of navigation constraints that we permit: incorporating overconstraints during solving to prune solution space; dealing with semi-algebraic sets, that is, dealing with polynomial inequalities during solving. The solver [49] satisfies all of these requirements.

6.1.4 Graphical User Interface (GUI)

Interaction with the user through the GUI is central to the ESM method, and benefits from a well-designed GUI. The GUI requirements however are easy to describe: adequate text and menu interaction as well as 2D and 3D canvases to enable the following. The GUI is used to input: the constraint system and a conceptual decomposition or feature hierarchy, (these could have been partially solved and stored from an earlier session), any updates to these, and interactive user input during the constraint parsing and equation building stages

and realization space navigation. The GUI is also used to modify the object, constraint and feature repertoire or constraint-to-equation parse tree. The GUI is used to output: the DR-plan of the input constraint system consistent with the input conceptual decomposition, the system of equations as they are being parsed in stages from the constraints, the subsystem solutions corresponding to the nodes of the DR-plan, a realization of the partially solved system as the navigation proceeds, and the final realization. A good example is the GUI implemented in FRONTIER [15, 50] and shown in the figures in Section 6.2.

6.2 Navigating the Solution Space

This section has two parts. The first describes the overall *Equation and Solution Manager (ESM)* navigation method and the common, augmented DR-plan datastructure - used by this method and its companion methods - that aids efficiency. The second part describes its integration into a standard constraint solver architecture (Figure 6–1) specifically, the dataflow, communication with the companion methods that satisfy the requirements given in Section 6.1, and a sketch of its implementation in the FRONTIER constraint solver [15].

6.2.1 The Basic ESM Method

The ESM takes as input the constraint graph, the DR-plan output of *any* DR-planner as in Section 6.1; a stable and combinatorially optimized system of constraints for each cluster in the DR-plan - that is, the output of *any* combinatorial cluster-system optimizer as in Section 6.1; and finally navigation constraints, such as redundant or consistent overconstraints and chiralities. It generates an algebraic system of equations and inequalities for each cluster. Then, using its access to *any* algebraic-numeric solver as in Section 6.1, it offers the user a tractable, interactive visual walk-through of the solution space by outputting to the GUI - and storing in the augmented DR-plan datastructure - the partial realizations or solutions of the cluster subsystems in an efficient manner.

Interactive input to the method includes the user's choice of solutions to the clusters and additional navigation constraints added on by the user, using which the ESM method recursively assembles the desired solution of the entire system.

The method provides fully flexible backtracking for redoing the user's choices starting from *any* cluster. Finally, the method can be run fully automatically, without any user intervention, to output a realization of the entire constraint system, including navigation constraints that are input apriori.

The Solveforest routine can be merged into the Solvecluster routine. It is not necessary, except for clarity, and for dealing with the case where the DR-plan has several roots - in case the input constraint system was underconstrained.

Solveforest (set S of clusters)

```
    [1) At top level of recursion S represents
    a complete decomposition of given system into
    maximal, well-overconstrained subsystems.
    2) At lower levels of recursion, S represents
    an optimal covering set output by CCO]
For each cluster C in S,
Solvecluster(C)
Return user-chosen solution U(C) for each cluster C in S.
   [efficiently stored in an augmented DR-plan
    datastructure, described below]
```

Solvecluster (C)

```
    [The input is CCO's output for C:
    1) at bottom level of recursion, a well-constrained
    system involving primitive geometric elements;
    2) at higher levels, anoptimal covering set S of
    solved child clusters Ci, and a stable, independent
    set of constraints between the Ci's, partitioned into
    tree overlap constraints and  non tree constraints]
Solveforest (set S of Ci's)
Repeat
    CEP(C)
        [Gets from user additional navigation constraints
         for C.
         1) In some iterations, especially the first,
```

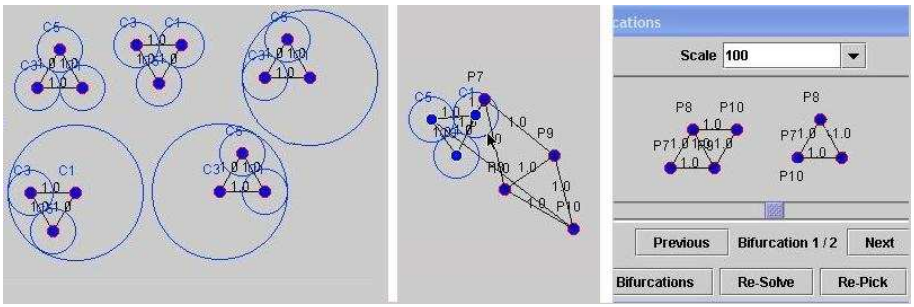

Figure 6–2: Solutions of the cluster formed by the 3 circles (Figure 1–1).

```
        it creates algebraic system Alg(C)
        for solving C, using the user-chosen solution for
        each Ci and the non-tree constraints in the stable
        set of constraints between the Ci's,
        output by the CCO.
        2) In some later iterations, it prunes any already
        existing set Sol(C) using the most recently input
        navigation constraints.
        3) It Interacts with user further for massaging
        this algebraic system to reduce complexity]
    Algebraic-Numeric Solver(Alg(C))
        [Returns solution set Sol(C) for C: rotation
        of each Ci w.r.t. its parent in the
        tree of overlap constraints given by CCO for C.
        The coordinate system of C is the same as
        the home or root cluster of the tree
        of cluster overlap constraints  output by the CCO:
        the home cluster's
        rotation/translation is fixed to be
        the identity.]
    Communicate the set Sol(C) visually to user via GUI
Until user is satisfied and picks a solution U(C)
Return user-chosen solution  U(C)
```

This visual walk-through is an interactive process, permitting the usual 3D visual tools such as panning, zooming, rotation to visualize the various solution possibilities, which cannot be fully illustrated by the figures here. The interested reader is encouraged to download and run the (opensource) software [15] (Figures 6–2, 6–3, 6–5, 6–6, 6–7).

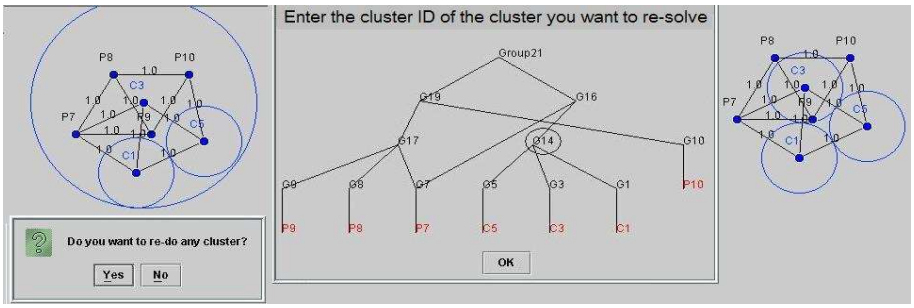

Figure 6–3: Complete solution of root cluster (Figure 1–1) after choosing subsolutions.

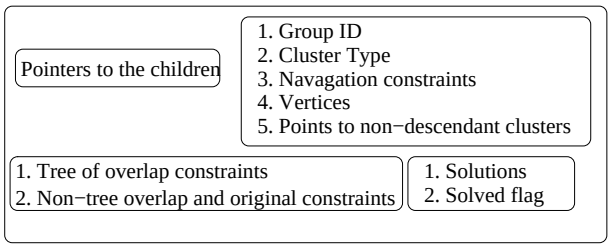

Figure 6–4: The structure of the cluster.

A cluster object represents a DR-planner's simplification [2, 6, 9, 10, 11, 15] of a well-constrained or a well-overconstrained subgraph, as well as its original subgraph.

Figure 6–4 shows the contents of the cluster.

1. *Hierarchy field.* These cluster objects are hierarchical and contain pointers to their sub-DR-plan, a directed acyclic graph (DAG), i.e, the DAG interrelationships are stored within each cluster C as a list of its immediate children C_i.

2. *Combinatorial descriptive fields.* The cluster C contains more descriptive fields, 1) a group ID corresponding to a feature, if any, that corresponds to this cluster C in the input feature hierarchy. Note that not every feature in the input feature hierarchy is a cluster; however, every cluster feature must appear in the output DR-plan, if the DR-planner satisfies the desirable feature incorporation property discussed in Section 6.1. 2) Another important descriptive field is the cluster type that indicates whether this cluster is well-constrained,

well-overconstrained (combinatorially), and a list of reducible overconstraints directly associated with C. , that is, does not lie within any of the child clusters of C. 3) Another field contains other navigation constraints that are directly associated with C (these are discussed later in this section). 4) Another field is a list of original vertices or geometric primitives that constitute this cluster. 5) In the case of module-rigid DR-planners [12], as mentioned in Section 6.1, some clusters become clusters only after other (non-descendant) clusters have been solved, inducing a solving priority order. Thus a cluster contains pointers to such non-descendant clusters.

3. *Fields giving input/output to CCO.* The cluster C stores the input to and the output of the CCO companion method, that is, a complete set of maximal rigid clusters within C, of which a subset have been picked by the CCO as the children of C. The cluster structure contains a stable, independent system of cluster overlap and original constraints between the (primitive elements of) children of C. These are partitioned into two parts, 1) a tree of overlap constraints, with a home or root cluster and a set of non-tree overlap and original constraints. 2) The original constraints include all non-reducible constraints in C.

4. *Fields describing algebraic solution.* The cluster object has several fields that store the solved degrees of freedom of the cluster. A list of strings is stored, one for each solution of the cluster returned from the algebraic-numeric solver (given fixed chosen solutions to each of the child clusters). When the user selects a desired solution, that string is parsed into a list of actual degree of freedom values for this cluster. Multiple copies of the list are permitted, one for each parent of the cluster in the DR-plan, if they exist. Note that especially when the DR-plan incorporates an input feature hierarchy or partial decomposition, the DR-plan could be a true DAG with child clusters shared by several parent clusters. The final field is a flag that indicates whether the current Cluster C has been solved or not.

A crucial feature of the datastructure that maintains clarity and efficiency during navigation and backtracking is that each realization of the cluster C is not stored as positions of each geometric element in the cluster - this would cost linear time and space for each child cluster - but rather it is stored as the homogeneous rotation/translations (a constant number of matrix entries) of C's child clusters that resolves the constraints between them. The constraints are the non-tree portion of the independent set of constraints output by the CCO companion method and the rotations are associated with the overlap tree output by the CCO companion method, expressing a child cluster C_i's rotation with respect to another child cluster C_j of C, namely C_i's parent in the tree of overlap constraints output by the CCO companion method. Only when the position of a primitive geometric element v in a cluster C is needed is it actually computed. This could be needed either by the *constraint equation processor (CEP)* of the ESM method described next, in order to construct a stable system of equations for C's parent in the DR-plan, or to prune the solution set of C, or when a particular realization of C needs to be displayed to the user.

The position of a primitive element v in C is obtained by taking a nested product of rotations. (i) The "outer" product is a product taken over any of the paths that lead from the current cluster to a leaf/sink of the DR-plan that corresponds to that geometric element. This path is not unique if the geometric element v is shared by more than one cluster appearing in the DR-plan, but all these paths provide the same, unique position for the geometric element v: this is ensured by the CCO companion method. (ii) Each "inner" product is associated with a single cluster D along the above path: it is a product of rotation matrices that place D within the coordinate system of its parent E in the DR-plan. That is, it is the product of rotations along the unique path to D from the root or home

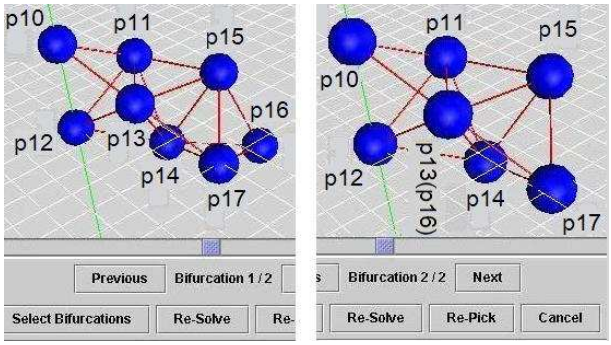

Figure 6–5: Navigating by bottom-up traversal of G79 in DR-plan (Figure 1–4).

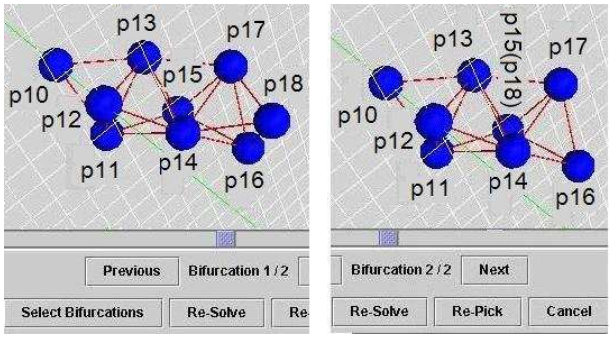

Figure 6–6: Navigating by bottom-up traversal of G82 in DR-plan (Figure 1–4).

cluster - in the tree of overlaps for E generated by the CCO companion method (Section 6.1).

Thus the entire DR-Plan object is simply a pointer to the root node of a list of clusters as they are described above. In case the graph is underconstrained, the root node is a dummy node, whose children are the actual sources of the DR-plan, i.e, a complete set of maximal clusters (Section 6.1).

The CEP method is a crucial part of the ESM method and has the following functionality: It generates a system of equations and inequalities for solving a cluster C, using (a) the output of the CCO companion method; (b) the user-chosen solutions for the child clusters C_i of C; and (c) navigation constraints directly associated with C, including reducible overconstraints which have been isolated by the DR-planner. In order to obtain algebraic systems of low complexity, it helps to permit user interaction during the process of building the system corresponding to

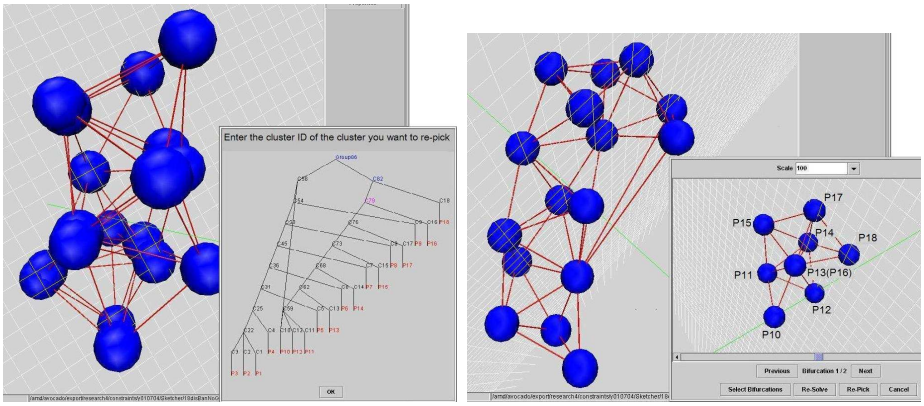

Figure 6–7: Complete solution of constraint system (Figure 1–4).

a cluster. The CEP is additionally "editable" since it permits modifications to the constraint repertoire as well as the constraint parsing mechanism to be *input* by the user systematically and efficiently instead of having to rewrite the code for the ESM. The CEP's user interaction is effected via the GUI text interface, discussed in the communication and integration section.

The CEP has a two stage operation. The first *conversion stage* converts edges or constraints in the *non tree* portion of the output of the CCO into algebraic equation strings understandable by the AlgebraicSolver. These equations are between variables describing the positions of the primitive elements in the child clusters C_i of C. The second *substitution stage* of the CEP expresses the position variables corresponding to a primitive element v in a child cluster C_i in terms of the *variable* rotation and translation of the cluster C_i within C's coordinate system obtained from the *tree* portion of cluster overlaps returned by the CCO.

The CCO's properties described in Section 6.1 guarantee that the resulting system from the conversion and substitution stages is a stable, independent system of algebraic equations for solving C.

Conversion stage of the CEP proceeds in steps, so that at any step, the user can interfere and make substitutions that make the system of equations simpler. The heart of the CEP is a datastructure and look-up system, the

constraint-to-equation parse tree (*CEP tree*), which takes the unique characteristics of each edge or constraint and searches an index of user-defined equations strings for a match. The CEP organizes those strings into a n-ary tree representing a natural hierarchy of constraints - this conceptual hierarchy is also mirrored in the representation of constraints used by any standard constraint solver GUI. The use of this tree data structure makes the searching of the constraint list, and conversion of constraint to equation both efficient and more interactive. Moreover, this datastructure can be input or modified by the user, making it fully editable and extensible, as discussed below.

The CEP tree is organized by equation classes and sub-classes. For instance, consider a distance constraint between two points and a tangency constraint between a two circles. The distance has the equation $(x1 - x2)^2 + (y1 - y2)^2 = d^2$, and the tangency has the equation $(x1 - y2)^2 + (y1 - y2)^2 = (r1 + r2)^2$, where $x1, x2, y1$, and $y2$ are either the locations of the points or the centers of the circles, d is the distance between the points, and r1 and r2 are the radii of the two circles. These equations fall into the same equation class because they are in a sense generated by the same basic template or "grammar." In this example, the tangency equation would be a sub-class of the distance equation since its equation could be created with the substitution of the d variable in the distance equation with the string $(r1 + r2)$. This relationship would be represented in the CEP tree by two nodes, one node for each equation form and the tangency node would be a child node of the distance node. To complete the data structure, each node stores (a) a list of edge objects which map to an equation and (b) either an equation or a list of substitutions which create the new child equation from the parent equation. By storing only the substitution information, the process of converting the list of constraints to a list of equations is highly simplified.

When the equation system for a current cluster C is being built one equation at a time, that is, when single constraints are parsed one at a time, the actual variable names occuring in the output equation are irrelevant during the conversion stage - they will be appropriately modified during the substitution stage described below.

However, the process of parsing a single constraint can be extended to parsing a system of constraints en bloc. This requires more care with preserving identical and distinct variable names that appear in different constraints. Generating the corresponding equation set involves traversing several paths of the parse tree in lock step, with substitutions being made along the way. The user can interfere at any stage and make substitutions to simplify the entire algebraic system. The user interaction proceeds through a simple GUI text interface discussed in the communication and integration section below.

Substitution stage of the CEP re-expresses the position variables - corresponding to primitive element v in a child cluster C_i - which occur in the equation system obtained after the conversion stage. These variables are expressed in terms of (a) the *variable* rotation and translation of the cluster C_i within C's coordinate system obtained from the *tree* portion of cluster overlaps returned by the CCO. and (b) the already solved, *constant* position of v within C_i's local coordinate system. Both (a) and (b) require products of rotation matrices, as in the description of the efficient, augmented DR-plan datastructure. In the case of (b), a nested product is required, with the "outer" product taken over the clusters on any path in the DR-plan from C_i to v. In the case of (a), there is only an "inner product" of the rotation matrices along the path to C_i from the home or root cluster - in the tree of overlap constraints for C returned by the CCO companion method.

Navigation constraints are used not only to prune the search during user directed navigation, but more importantly, to convert the ESM method into a

(tractable) automatic search for desired solutions for any cluster of the DR-plan. These constraints can either be given apriori or interactively during a user-directed navigation, to prune unwanted realizations of the DR-plan's clusters encountered thus far.

Leveraging the modularity of the ESM method discussed above - that is, its separation from the DR-planner, and its applicability to *any* input DR-plan as in Section 6.1 - permits methods developed along with other DR-planners for incorporating navigation constraints - to be transferred *practically unchanged* to the FA DR-planner and the ESM methods discussed above, for example relational and engineering constraints for triangle decomposable systems given by Fudos and Hoffmann [42]. We describe the two other common types of navigation constraints below.

One type of constraint used for navigation or picking out desired solutions for any cluster C of the DR-plan are simply redundant or overconstraints directly associated with C, that is, connecting primitive elements that lie in different child clusters of C. These are isolated by any DR-planner as in Section 6.1. There is a well-developed theory of unique realizations and so-called rigidity circuits obtained, for isolating desired solutions from the solution space using redundant constraints [30].

The other type of common navigation constraints are chirality [4, 41], also called order type or relative orientation constraints. Chirality constraints can be infered from the input sketch. For example, for point objects, these constraints assign signs to the $D+1$ by $D+1$ determinants of the homogeneous coordinate matrices obtained from each set of $D+1$ point objects. I.e, these constraints are $D+1$ degree polynomial inequalities. These correspond to a complete oriented matroid specification [51]. Alternatively, a *partial* oriented matroid specification could be specified by the user: these assert intersection or separation

of pairs of convex hulls of subsets of point objects (called respectively circuits and co-circuits in oriented matroid terminology). For example, two line segments could be constrained to intersect, or a point could be forced to lie in the convex hull of some other set of points. These, too could be written as polynomial inequalities. Chirality constraints, like redundant overconstraints, are also directly associated with a unique cluster C in the DR-plan: the smallest cluster that contains all the primitive elements that participate in the chirality constraint. This also implies that not all of the participating primitive elements fall in any single child cluster of C.

The navigation constraints associated with a cluster C are processed by the CEP using the same Conversion and Substitution stages described above. At the end of these stages, the navigation equations and inequalities are not in terms of the primitive elements that they involve. Instead, they are in terms of the rotation matrix entries (variables) that place the children C_i of C (in the DR-plan), within C's coordinate system. Thereafter, during each iteration in which the CEP is called for cluster C, they are dealt with in one of two ways. If the algebraic system associated with the cluster has not yet been solved (in the first iteration), or the number of solutions is beyond some predefined threshold (in later iterations), and the AlgebraicSolver is capable of handling redundant equations and polynomial inequalities, then these processed navigation equations and inequalities are tacked on to the stable, independent system for C generated as described above, and sent to the AlgebraicSolver. If not, i.e, if either of these conditions does not hold, these navigation equations and inequalities are simply used to prune the solution set for C. Note that since the navigation equations and inequalities have already been processed by the Substitution stage of the CEP, this pruning can be done by directly using the solutions output by the AlgebraicSolver, i.e, using the rotation

matrix entry values for the children of C. No computation of the actual positions of the primitive elements in C is required.

The design of the CEP permits the user to flexibly *create* and *modify* the parse tree described above, without hardcoding it.

In creating the parse tree, the first node is a child of a pre-existing dummy node, Root. Newly created nodes can be the children of any previously entered node or Root. After the user has chosen the position of the node within the forming tree, all other necessary information, such as the new label and equation, are obtained. The final step in node creation is to attach a list of the constraint types that will be indexed to this new node. The user input a text list of all the constraint types which map to this node. The user repeats the process of node creation until the entire CEP tree has been entered. The user then saves the entire CEP tree to be used for future parsing.

6.2.2 ESM's Communication with User and Integration

The standard architecture of a geometric constraint solver together with the ESM module is given (Figure 6–1).

Next we discuss the details of the dataflow when the ESM is integrated within such a standard architecture, including communication with the user. We additionally sketch FRONTIER constraint solver's implementation of these aspects [15].

The ESM must continually interact with the user to select various subsystem solutions and to direct the path towards final solution. All communication between the modules takes place via the UTU or the universal transfer unit, which is also the main driver of the back-end modules. While there are numerous types of information the ESM can pass back to the GUI, their process is always the same. First, the current state of the DR-Plan and all additional communication information is sent to the GUI via the UTU. The GUI reads only the communication

information, but stores the residual datastructure information while it processes the communication information. The residual data structure information is passed back to the UTU, if necessary, during the next request, and using it, the UTU can rebuild the current state of the solution process and the ESM can continue where it left off. The detailed actions of the GUI, UTU, and the ESM in each mode of communication during the solving phase are outlined in the next sections.

Once the DR-planning phase is over and the DR-plan has been displayed to the user, and when the UTU receives the flag indicating that the user has selected the user-interactive Navigate option (as opposed to the Autosolve option), the UTU simply sends the constraint graph and DR-Plan directly to the ESM. When the ESM receives the continue flag, it begins the solution-navigation process and the visual walk-through or steering through the solution space as explained in Section 6.2. The ESM traverses the DR-plan bottom up, as explained above, until it reaches the point where it has several cluster solutions to offer to the user. At this point, the user must select one of several possible solutions for the cluster that the ESM will use to create the final solution for the entire graph. This information must be passed from the ESM back to the Sketcher for display. The ESM sends back two lists of information via the UTU. First, for each solution possibility, the ESM writes out a list of the primitive elements in the cluster. For each element, the ESM writes the object's ID, and a list of its solved position variables. Second, the ESM writes a list of the clusters contained in the current DR-Plan and a boolean flag to indicate whether they have been solved or not. This information allows the user to locate where the solving process is w.r.t. the DR-Plan when making his/her choice of a solution. As a final step, the ESM sets a flag to indicate that the information being sent is a new solution possibility and returns.

The Sketcher displays the solution possibilities to the user, one at a time, in a separate window. The original sketch and a *partially solved sketch* as in Figures

6–2, 6–3, incorporating the subsystem solutions chosen so far into the unsolved original sketch are also displayed and allows the user to make a choice of several options described below.

1. Select a solution and continue solution - In this case, the Sketcher simply sets the ESM communication flag to indicate a solution choice and sends the UTU the number of the chosen solution. The ESM uses this to updates the DR-Plan to contain the rotation matrix values for the chosen solution and continues the solution navigation process. Assuming that the user always selects a solution, this process of solving, solution selection, and DR-Plan update would continue until all the subsystems in the DR-Plan have been solved.

2. Choose to backtrack - If the user decides that none of the current solution choices are appropriate, they may choose to edit their earlier selections. For editing, the user has two options: resolve the current cluster with new choices for its children's solutions, or repeat the solving of some previous cluster after making new choices for the solutions of its children. In either case, the Sketcher sends a flag back to the UTU that indicates that backtracking is to occur and an ID of the cluster from which the users wishes to continue solution. The UTU edits the DR-Plan and resets the "solved" flags that indicate that that cluster is the next to be solved. The ESM will ignore any previous solutions if the fin flag has been unset, and when the UTU calls it with the altered "solved" list, it will continue solving from an earlier point exactly as described above.

3. Choose to update - This final option permits the user to update the input constraint graph and feature hierarchy, because a subsystem is found to have either zero or infinitely many solutions (due to generic or nongeneric constraint dependencies undetected by the DR-planner). The user believes there is something wrong with the input graph itself, rather than the bifurcation choices that have been made earlier. Many modifications of the graph do not require repeating

the DR-planning of the system. These changes can be made interactively during solution navigation process and much of the original DR-plan and solutions can be reused. This is one of the advantages of the Frontier vertex based DR-planners [2]. The the algorithms, communications and dataflow for such updates are described by Sitharam [11].

Assuming a solution to the current system exists and the user can select subsystem solutions to find it, all communication ends when all the root clusters in the DR-Plan have been solved and the user does not wish to backtrack. At this point, the ESM writes a flag to indicate that a final solution is to be returned. Then it returns a ordered list of all objects, their ID's and the final positions of their degrees of freedom.

If the user chose the autosolve as opposed to the navigate option, there is no interaction between the user and the ESM after the DR-Plan has been accepted. The process of auto-solving is the same as the navigate solution option except that when the ESM would request a realization choice from the user, the ESM automatically chooses the first solution that satisfies the navigation constraints. Backtracking automatically occurs when no solution is found for a cluster C for the current choices of the solutions for the children of C. At the end of the auto-solution process the ESM either returns a final solution exactly as above or it returns a flag that indicates that no valid solution exists.

The CEP permits the user to both set up the CEP tree, and to interactively build subsystems of equations that the ESM sends to the AlgebraicSolver. Both types of communication take place via the UTU, and through the GUI. The GUI opens a text window for this communication.

The GUI options allow the user to set interactive/automatic mode for the CEP at any point during a session. In the interactive mode of the CEP, the user is again given the option of taking the Entry mode (where the user can edit the parse

tree, add new types of constraints and so on,) or the Parse mode (for the navigate option of the solving phase). In the latter case, the user is prompted each time when a subsystem of equations is built by the ESM to send to the AlgebraicSolver. If the autosolve option is chosen, the default choice is to shut off all user interaction with the CEP.

The GUI, sometimes just refered to as Sketcher, is written in Java, Java 3D and is the main driver for FRONTIER and is used for all input and output. The DR-planner, CCO, UTU, ESM together with CEP, and so on, are written in C++. The C++ driver is the UTU, which also coordinates the communication. The AlgebraicSolver is off-the-shelf [49] and not discussed here.

Implementing the Communication between the Modules The DR-plan datastructure is physically shared by the C++ modules, and is communicated, through the UTU, to the Java modules using a Java Native Interface (JNI arrays); however the same datastructure is conceptually mirrored in the object oriented hierarchies used by the GUI's Java modules. The GUI is used for all user input and for all output displays. All information passed between the GUI and the backend modules (C++) is passed through a JNI interface of three arrays, one of integers, one of doubles and one of characters. The GUI writes information into these three arrays, and natively calls the C++ code with these three arrays as parameters. The C++ driver, the UTU, parses these arrays and from them creates (or restores) the augmented DR-Plan data structures described earlier. The UTU then directs the ESM to accomplish the specific request made by the user, described above, in the Dataflow section. The different requests are indicated by the use of an integer flag that is passed as the first integer in the integer array.

6.3 Conclusions

We list below the main features of ESM:

The ESM method *recursively assembles* the desired solution of the system G: it *prunes* the solution space of G by eliminating solutions that are inconsistent with the user's interactive solution choices for well or well-overconstrained subsystems S of G. These are selected by *visually walking the user through* all the solutions - for a current subsystem S - that are consistent with his/her previous solution choices for subsystems *of* S. These subsystems S are taken from a DR-plan of G. When this DR-plan is in correspondence with a design decomposition of G, the ESM method provides the user conceptually meaningful control. The ESM method is typically *fully scalable* in that only a small (constant, independent of $|G|$) number of solution choices are offered to the user at any stage, in a tractable (linear in $|G|$) number of stages.

The ESM method permits the user to *backtrack* and change his/her previous solution choices at any stage. It also permits additional *navigation constraints* and other inputs by the user that help reduce the solution choices for the current subsystem S, or help speed up the solving of S. These could be provided apriori or interactively. The CEP permits the user to edit the repertoire of navigation constraint types and the manner by which they are parsed into algebraic equations for efficient solving. Alternatively, the entire ESM method can be switched to *automatic*, that is, requiring no user interaction. In this case, its efficiency and scalability depend on the tightness of the apriori input navigation constraints.

Given access to the companion methods in Section 6.1, the modular ESM method is original, but conceptually simple. It involves significant interaction with the user and with the companion methods. Hence the challenge lies in its clean integration into the geometric constraint solver architecture (Figure 6–1) [10], leveraging and naturally extending the data structures and data flow. The original contribution of this chapter is represented by the dotted balloon in Figure 6–1. A complete implementation (Figure 6–1) can be found in the FRONTIER opensource

software [15], which is to date the only complete 3D geometric constraint solver available.

CHAPTER 7
COMBINATORIAL OPTIMIZATION OF ALGEBRAIC COMPLEXITY

In Chapter 7 we show that, Most major geometric constraint solvers use a combinatorial or graph algorithm to generate a decomposition-recombination (DR) plan. A DR plan recursively decomposes the system of polynomial equations into small, generically rigid subsystems that are more likely to be solved by algebraic-numeric solvers.

Especially for 3D geometric constraint systems, a further optimization of the algebraic complexity of these subsystems is both possible, and often necessary to solve the well-constrained systems selected by the DR-plan. To attack this apparently undocumented challenge, we use principles of rigid body transformation and quaternion forms and we combinatorially optimize a function over the minimum spanning trees of a graph generated from DR-plan information. This approach follows an interesting connection between the algebraic complexity of a well-constrained system and the topology of its maximal well-constrained proper subsystems. The optimization has two secondary advantages: for navigating the solution space of the constraint system and for mapping solution paths in the configuration spaces of the subsystems.

We compare the reduction in algebraic complexity of the subsystem after optimization with that of the unoptimized subsystem and illustrate the practical benefit with natural examples that could only be solved after optimization.

7.1 An Instructive Example

In this section, we show three increasingly sophisticated and increasingly efficient approaches to *representing the active constraints* needed to resolve a cluster of the DR-plan satisfying the properties of 1.2. Since we want to be able to

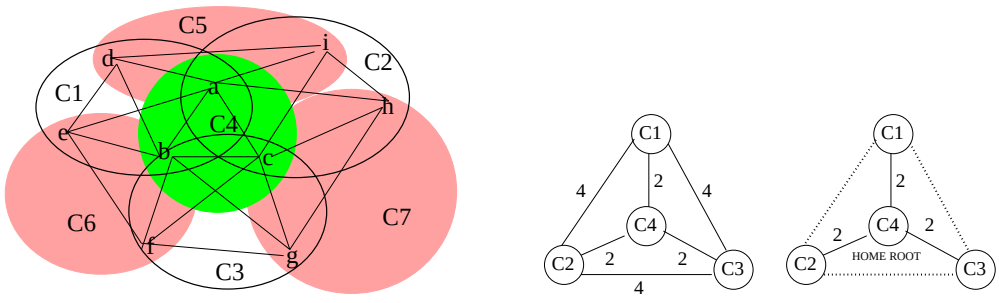

Figure 7–1: The child clusters of `hex_tet` and the weighted overlap graph.

compute any and all solutions, the number of variables and the algebraic degree of the constraints plays a crucial role.

Figure 7–1 shows the constraint structure and child clusters C_i of a problem we call `hex_tet` , for reasons that will become obvious when we look at specific geometric instances later on. All vertices represent points with 3 degrees of freedom (dof) and all edges are distance constraints removing 1 dof each. The child clusters C_1, C_2 and C_3 are tetrahedra and all other child clusters are triangles. The actual distance values are not yet relevant for the discussion in this section (but will be in Section 7.3 when we actually solve instances). We may think of them as equal (except for the 3 distances ef, gh and di which need to be larger in order for a solution to exist).

The cluster C satisfies the DR-plan properties of Section 1.2. Specifically, the child clusters form a complete set of maximal clusters properly contained in C, and their pairwise intersections are trivial. Minimal covering sets are: $\{C_1, C_2, C_3\}$, $\{C_5, C_6, C_7\}$.

In the following, we denote by

$$\mathbf{x}_{a,C_i} := \begin{bmatrix} x_{a,C_i} \\ y_{a,C_i} \\ z_{a,C_i} \end{bmatrix} \in R^3$$

the coordinates of point a in C_i's local coordinate system and by

$$\mathbf{x}_{a,C,C_i} := \begin{bmatrix} x_{a,C,C_i} \\ y_{a,C,C_i} \\ z_{a,C,C_i} \end{bmatrix}$$

the coordinates of point a *parameterized* in the dof's in C's coordinate system.

In particular, $\mathbf{x}_{a,C,C_i}$ may depend on rotation angles that will only be resolved to a final position $\mathbf{x}_{a,C}$ in cluster C by solving the constraint system.

7.1.1 The Unoptimized Polynomial System

Current constraint solvers, such as FRONTIER [15], construct the polynomial system corresponding to `hex_tet` as follows:

Pick an arbitrary minimal covering set of child clusters. For the example, we choose $\{C_1, C_2, C_3\}$. Set the coordinate system of C to one of the child clusters. This child cluster is called the *home cluster* for C. For the example, we choose C_1.

Each of the other two clusters' position and orientation within C's coordinate system can be expressed using a composite 3D rotation matrix $\mathbf{M}$ and a translation vector $\mathbf{t}$. For example, point i in C_2 can be written as

$$\mathbf{x}_{i,C,C_2} = \mathbf{M}_{C_2}\mathbf{x}_{i,C_2} + \mathbf{t}_{C_2}$$

where $\mathbf{t}_{C_i} \in \mathbf{R}^3$ is a translation that positions C_i in C's coordinate system and

$$\mathbf{M}_{C_i} = \begin{bmatrix} s_2 s_3 & s_2 c_3 & -c_2 \\ s_1 c_2 s_3 - c_1 c_3 & c_1 s_3 + s_1 c_2 c_3 & s_1 s_2 \\ s_1 c_3 + c_1 c_2 s_3 & -s_1 s_3 + c_1 c_2 c_3 & c_1 s_2 \end{bmatrix}$$

is a composition of 3D rotations that orient C_i in C's coordinate system. The entries of $\mathbf{M}_{C_i}$ are polynomials of total degree at most 3 in the six variables $s_{1,C_i}, s_{2,C_i}, s_{3,C_i}, c_{1,C_i}, c_{2,C_i}, c_{3,C_i}$. These variables represent the sines and cosines of the rotation angles and are therefore related by three *trig-relation* equations of the form $s_i^2 + c_i^2 = 1$.

Since C_2 shares the point a with C_1 and the coordinates of C_1 are fixed, $\mathbf{t}_{C_2} = \mathbf{x}_{a,C_1} - \mathbf{x}_{a,C_2}$. Similarly, $\mathbf{t}_{C_3} = \mathbf{x}_{b,C_1} - \mathbf{x}_{b,C_2}$.

This leaves six active constraints for C: three distance constraints of type

$$\delta^2_{i,C_j,C_k} = \|\mathbf{x}_{i,C,C_j} - \mathbf{x}_{i,C,C_k}\|^2$$

where δ is the prescribed distance and three overlap constraints

$$\mathbf{x}_{\ell,C,C_j} = \mathbf{x}_{\ell,C,C_k}, \qquad j,k \in \{1,2,3\}.$$

Thus, to resolve C, we need to solve a system of *twelve polynomial equations of maximum degree 6* in twelve variables s_{i,C_j} and c_{i,C_j}, $1 \le i \le 3$ $2 \le j \le 3$. The six trig-relations have degree 2. The three distance constraints each have a total degree 6 (since, for example, the rotation matrix results in cubic expressions for the value of x_{i,C,C_2}) and the overlap constraints each have total degree 3.

7.1.2 Using Quaternions

A more careful approach to step 3 of the procedure can reduce the degree of the equations. Each of the two clusters C_2 and C_3 share a point with the home cluster. Then their position and orientation within C's coordinate system can

be represented as a rotation about an arbitrary axis pivoting through the shared point.

It is well-known that rotation about an axis $\mathbf{v} \in R^3$ by an angle α can be expressed as

$$\mathbf{q}^{-1} \cdot (0, \mathbf{x}) \cdot \mathbf{q}, \text{ where } \mathbf{q} := (\cos(\alpha/2), \sin(\alpha/2)\frac{\mathbf{v}}{\|\mathbf{v}\|}).$$

Here $\cdot$ is the quaternion multiplication. In fact, any rotation matrix can be expressed in terms of a (unit) quaternion as

$$\mathbf{Q} := \begin{bmatrix} 1-2q_2^2-2q_3^2 & 2(q_1q_2+q_0q_3) & 2(q_1q_3-q_0q_2) \\ 2(q_1q_2-q_0q_3) & 1-2q_1^2-2q_3^2 & 2(q_2q_3+q_0q_1) \\ 2(q_1q_3+q_0q_2) & 2(q_2q_3-q_0q_1) & 1-2q_1^2-2q_2^2 \end{bmatrix}, \qquad \sum_{i=0}^{3} q_i^2 = 1.$$

For example, the rotation about x-axis by $-\alpha$ is represented in terms of $q = (c, s, 0, 0)$ where $c := \cos(\alpha/2)$, $s := \sin(\alpha/2)$ and hence

$$\mathbf{Q}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1-2s^2 & 2sc \\ 0 & -2sc & 1-2s^2 \end{bmatrix}, \quad \sin(\alpha) = 2sc, \cos(\alpha) = 1-2s^2.$$

Back to our example, in step **3**, we translate C_2 so that $\mathbf{x}_{a,C,C_2} = \mathbf{0}$, that is the point a that it shares with the home cluster becomes the origin. Then we apply to each point i the quaternion matrix $\mathbf{Q}_{C2}$ (with unevaluated q_i). Finally, we translate to the target position $\mathbf{x}_{a,C,C_1}$ of a in the home cluster:

$$\mathbf{x}_{i,C,C_2} = \mathbf{x}_{a,C,C_1} + \mathbf{Q}_{C_2}(\mathbf{x}_{i,C_2} - \mathbf{x}_{a,C_2}).$$

We are left with the three overlap and three distance constraints as in the previous formulation. However, we now need to resolve only *eight equations in eight variables of maximum degree 4*! Moreover, the three overlap equations are only quadratic, as are the two trig-relation equations for C_2 and C_3. The distance

equations di between C_1 and C_2, and ef between C_1 and C_3 are quadratic and only the distance constraint ih between C_2 and C_3 is of total degree 4.

7.1.3 An Optimized Alternative

Both the number of unresolved constraints and the degree of the equations can be further reduced. Consider the following construction for a constraint system expressing `hex_tet` . Choose $\{C_1, C_2, C_3, C_4\}$ as the (not minimal!) covering set. Choose C_4 as the home cluster. Without loss of generality, using basic trigonometry, the coordinate system for C is

$$\mathbf{x}_{a,C,C_4} = \mathbf{0}, \quad \mathbf{x}_{b,C,C_4} = \begin{bmatrix} \text{dist}(a,b) \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{x}_{c,C,C_4} = \begin{bmatrix} \text{dist}(a,c)c_{23} \\ \text{dist}(a,c)s_{23} \\ 0 \end{bmatrix}$$

where c_{23} and s_{23} are the sine and cosine of the angle between the edge a, b and the edge a, c.

Since C_4 and each C_k, $k = 1, 2, 3$ share two points, $p_k, \tilde{p}_k \in \{a, b, c\}$, the position and orientation of the cluster C_k are fixed except for rotation about the axis through the two points. The coordinates of an arbitrary point i in cluster C_k can then be represented as

$$\mathbf{x}_{i,C,C_k} = \mathbf{x}_{p_k,C} + \mathbf{R}_{C_k}(\mathbf{x}_{i,C_k} - \mathbf{x}_{p_k,C_k}).$$

where $\mathbf{R}_{C_k}$ is a linear expressions in 2 variables s_{C_k} and c_{C_k}. The variables represent the sine and cosine of rotation about $p_k\tilde{p}_k$ and satisfy $c_{C_k}^2 + s_{C_k}^2 = 1$.

This leaves only three distance constraints. To resolve the cluster C, we now only need to solve a system of *six quadratic constraints in the six variables* s_{C_j} and c_{C_j}, $1 \leq j \leq 3$ since all constraints, the three distance constraints and the three trig-relation constraints, are quadratic!

7.1.4 Interpretation

The reduction in algebraic complexity from the original to the optimized solution is significant. None of the algebraic and numerical solvers, we surveyed, succeeded in solving the original problem formulation but all solutions of the optimized system were obtained in less than one minute. Before we look at individual instances in Section 7.3, we characterize the principles underlying the optimized solution and give a general algorithm for optimization.

7.2 Optimizing Algebraic Complexity using Cluster Topology

In this Section, we formulate a combinatorial optimization of the algebraic complexity of the polynomial system for resolving a cluster C. We restrict ourselves to the 3D case – the 2D case is similar and simpler. Our definition of the optimization problem is accompanied by a sketch of an optimization algorithm.

We assume that the cluster C, for which the optimized polynomial system needs to be constructed, satisfies the DR-plan properties of Section 1.2. Specifically, we assume that its child clusters form a complete set of maximal clusters properly contained in a well-constrained cluster C; that all overconstraints and hidden constraint dependencies have been removed as required. We may also assume that the pairwise intersections of these clusters are trivial subgraphs since, if C is formed from a pair of child clusters whose intersection is non-trivial, the home cluster fixes the other clusters by fixing three of its points. We first define a natural method of describing the topology of the child clusters of a cluster.

Definition 7.2.1 *The overlap graph of a subset S of child clusters of C is an undirected graph whose vertices are the clusters in S. The edges represent pairs of clusters. If the overlap between the pair C_i, C_j is a trivial subgraph that reduces, after resolving incidence constraints, to k points then the weight of an edge is $w_{ij}(k) := 6 - 2k$.*

The weight of an edge (i, j) agrees with the number of rotation variables needed to represent the cluster C_i, if C_j were to be fixed as the home cluster. $\mathbf{R}$, the two variable rotation of C_i about the axis formed by the two shared points in the overlap; $\mathbf{Q}$, the four variable quaternion matrix representing rotation of C_i about the shared point in the overlap; $\mathbf{M}$, the six variable combination of rotation and translation of the unoptimized example.

Based on the examples of the previous section, we observe that (i) any choice of a set S of child clusters that forms a covering set for the parent cluster C, and (ii) any spanning tree T of the overlap graph of the clusters in S induces a system of equations for resolving C.

The choice H of a tree root represents the home cluster to be fixed. Since C is assumed to be wellconstrained, the number of variables and equations equals the total edge weight of the the spanning tree. Most sparse polynomial system solvers [52, 53] (geometric constraint systems are sparse) take time exponential in the number of variables. Hence the overwhelming factor in the algebraic complexity of the system is the number of variables. Degree and the number of terms are less important since the time complexity is polynomial in these parameters.

Algorithmically, optimization requires *finding a spanning tree of minimum total weight in the subgraph of the overlap graph induced by S, over all covering sets S*. Alternately, if S are chosen to be *minimal* covering sets, then the optimization requires *finding a tree of minimum total weight containing S, over all covering sets S*.

For general graphs and either arbitrary sets of vertices S of a fixed size or even for a fixed input set S, this problem is called the *Steiner tree* problem and it is NP-complete. Our overlap graphs, however, are special; the sets S are covering sets, and most importantly, since the cluster C is decomposed into its maximal clusters, the number of child clusters in most applications is no more than ten. So,

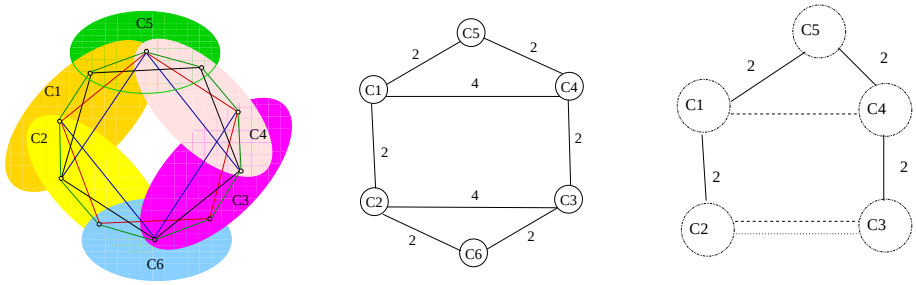

Figure 7–2: Constraint graph and weighted overlap graph of problem `pent_plat`

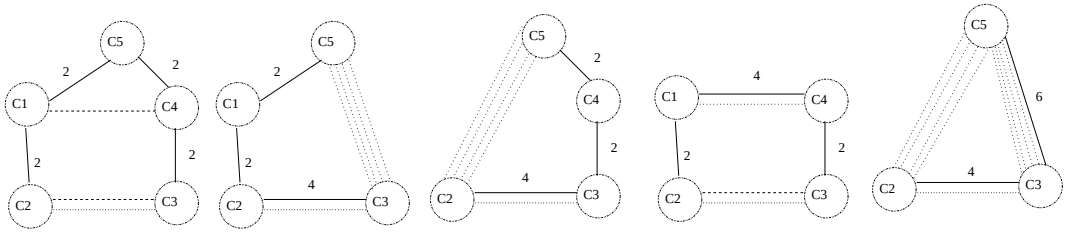

Figure 7–3: Five spanning trees of the covering sets of the example (Figure 7–2).

for our purposes, this problem is tractable if we use the original formulation in the previous paragraph, even if we exhaustively search through all covering sets S.

Figure 7–3 shows covering sets and trees for the example `pent_plat` (Figure 7–2) where child clusters $C_1, \ldots, C_6$ form a complete set of maximal child clusters of the parent cluster C. We are interested in those that have the minimum possible weight (in this case 8). Note the rightmost covering set only has a spanning tree of weight 10.

We can further optimize the degree among the set V_C of polynomial systems for C with minimum number of variables (and equations). The set V_C can be identified with the set of spanning trees for optimal covering sets S (that is, those that have spanning trees of minimum weight over all S). Now, following the examples of Section 7.1, given such an optimal covering set S and a minimum weight spanning tree T for S, the equations needed to resolve C are determined by (a) the cluster overlaps between pairs of clusters in S, corresponding to non-tree edges clusters in S, (b) distance constraints between clusters in S and (c) quadratic

trig-relation constraints relating the rotation parameters of each cluster in S except the home cluster.

The degrees of these equations are determined by the choice of the home cluster H or the root of T. To determine the polynomial degree of equations, let C_j be the direct predecessor of a cluster C_i on the path from the root to C_i. Then an overlap constraint involving a point in cluster C_i that is not in C_j has degree $\deg(C_i)$ at most

$$\deg(C_i) = \begin{cases} \max\{2, \deg(C_j)\} & \text{if } w_{ij} = 4, \\ \deg(C_j) + 1 & \text{if } w_{ij} = 2. \end{cases}$$

A distance constraint has, of course, twice the degree. Thus overall, a good intuitive heuristic is to use, as a root, a home cluster H that minimizes the depth of the rooted tree.

For the example trees (Figure 7–3) choosing any of the left 4 covering sets, we get 8 equations, but choosing the rightmost gives 10 equations. By choosing different home clusters say C_4 versus C_5 for the left-most tree we get different maximum degrees for the active distance constraint. The best choice overall is the left most tree with home cluster C_5, which yields 3 overlap constraints (between C_2 and C_3) of degree 2 and 1 distance constraint of degree 4.

The example shows that computing the degrees of the equations, that result for a given tree and home cluster, requires care. But the optimization problem and algorithm are straightforward:

over the spanning trees in V_C and the choices of their roots or home clusters H,

we *minimize the maximum degree of the system of equations* and,

in the resulting set of rooted trees V_{DC},

we *find those with the minimum number of equations of maximum degree.*

Keeping track of the spanning trees in V_C and V_{DC} can be achieved in polynomial time using the data structures [54]. However, in practice, the big

O complexity for creation and maintenance of these datastructures hides large constants. Thus for the small sizes of clusters and overlap graphs that are typical in most applications, an exhaustive algorithm, albeit of exponential time complexity works better.

7.2.1 Reduction of Algebraic Complexity through Optimization

The formal decrease in the *number of variables and equations* obtained through the optimization is easy to quantify. Without optimization, to solve a wellconstrained cluster C using k child clusters that form a minimal covering set, we would need $6(k-1)$ variables, independent of the number of nonempty overlaps between clusters. Using our optimization method, we would typically need two fewer variables for each point overlap and four fewer for each pair overlap. Let k_1 be the number of pair overlap edges, and k_2 the number of point overlap edges in a minimum weight spanning tree T of the overlap graph of the child clusters. Then we need at most $2k_1 + 4k_2 + 6(k - 1 - k_1 - k_2)$ variables. If $k_1 + k_2 = k - 1$, we reduce the number of variables by a factor linear in k, the number of child clusters.

The *total degree* of all the distance equations in the unoptimized system is 6. Let us assume the root in the minimum spanning tree T that maintains a distance of at most 2 (resp. 1) to each cluster involved in a distance constraint. Then, in the optimized system, the largest degree of a distance equation is at most 4 (resp. 2).

7.2.2 Removing Assumptions Made in Section 1.2

We turn to two assumptions made in Section 1.2. The first is to ensure that the system of equations constructed by the above algorithm is stable [13]. In the leftmost tree (Figure 7–3), although the cluster C is wellconstrained and the dof count is six, there are two non-tree edges representing overlaps. Choosing the overlap constraints between C_2 and C_3 would result in an independent system for solving C, but choosing the overlap between C_1 and C_4 would result in three dependent constraints. This problem is addressed independently and solved by

Sitharam et al. [13]. Here, we do not impose this constraint of stability into our optimization problem, since the feasible region for our optimization problem corresponds to choosing or describing an independent set – and this is nontrivial. We simply assume that any choice of the appropriate number of overlap constraints and other constraints is a valid choice.

The second assumption made in 1.2 is that there are no clusters (among the child clusters of C) whose clusterhood depends on other clusters. In fact, in order to obtain tractable, approximate characterizations of combinatorial rigidity [12], such dependent clusters are necessary. In our optimization problem, these dependent clusters have to stay with their partner cluster, affecting the set of valid choices for covering sets S. However, while outside the scope of this section, there is a relatively straightforward extension of the above optimization algorithm that would remove both of these assumptions.

7.3 Solutions to an Optimized System of Cluster Constraints

We now solve several geometric instances of the problem `hex_tet` from Section 7.1 and an instance of `pent_plat` from Section 7.2.

To be able to better judge the output of our calculations, we select symmetric setup for `hex_tet` : the three clusters C_1, C_2 and C_3 are initialized with identical *local vertex coordinates* $\mathbf{c}_{ij}$,

$$\mathbf{c}_{i,1} := \left[\begin{smallmatrix}-1\\-1\\-1\end{smallmatrix}\right], \quad \mathbf{c}_{i,2} := \left[\begin{smallmatrix}-1\\1\\1\end{smallmatrix}\right], \quad \mathbf{c}_{i,3} := \left[\begin{smallmatrix}1\\-1\\1\end{smallmatrix}\right], \quad \mathbf{c}_{i,4} := \left[\begin{smallmatrix}1\\1\\-1\end{smallmatrix}\right], \qquad i = 1, 2, 3.$$

The vertices are those of a regular tetrahedron and $\mathbf{c}_{ij}$ is the jth vertex of cluster i. The incidence and distance constraints that force the tetrahedra into a specific spatial arrangement are

$$\mathbf{c}_{i,2} = \mathbf{c}_{i+1,1}, \qquad \text{and} \qquad \|\mathbf{c}_{i,3} - \mathbf{c}_{i+1,4}\|^2 = \delta_i^2,$$

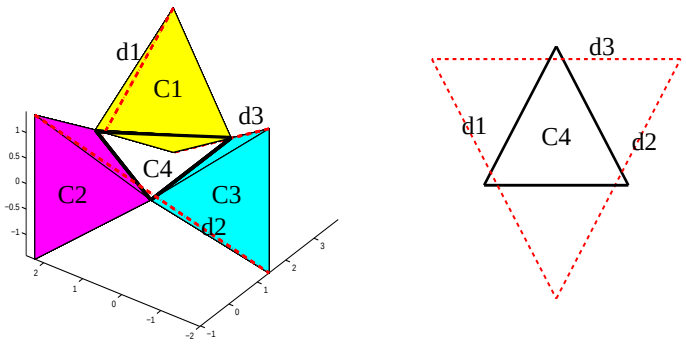

Figure 7–4: Unique solution to problem `hex_tet` .

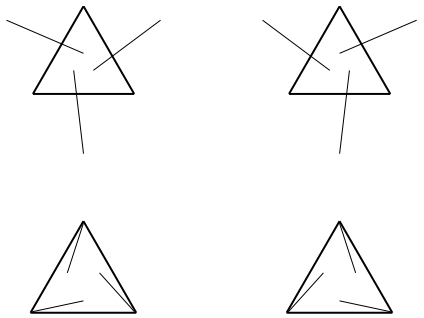

Figure 7–5: Diagramatic view of the four solutions to problem (**ii**).

where $i + 1 = 1$ for $i = 3$. The underlying geometry is visualized (Figure 7–4) for one, external choice of δ_i. We consider three cases. In each case, it was necessary to use the optimized formulation of the system described in Section 7.1 to be able to solve it with a numerical solver [49].

The maximal value for $\delta_1^2 = \delta_2^2 = \delta_3^2$ that still allows for a real solution is $\bar{\delta}^2 := 22 + 4\sqrt{2}\sqrt{3}$. Then there is exactly one solution (Figure 7–4). We juxtapose the spatial view with a more abstract, diagramatic view of the solution. In general, the spatial view does not yield a good visualization since the clusters intersect and block the view. Especially when presenting solutions, showing just the home cluster and the active distance constraints often reveals the underlying symmetries.

For $\delta_1^2 = \delta_2^2 = \delta_3^2 = \bar{\delta}$ (no square), we obtain four solutions with evident symmetries (Figure 7–5).

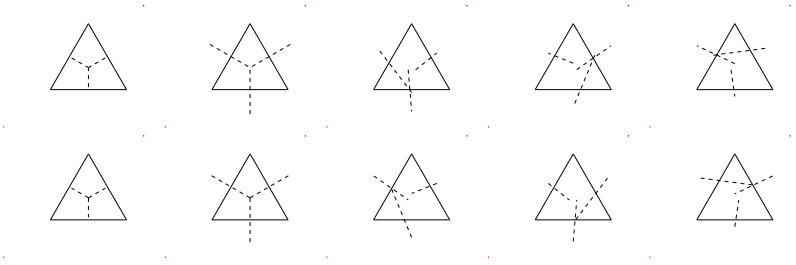

Figure 7–6: Diagramatic view of the ten solutions to problem (**iii**).

Finally, by altering the edge lengths of one cluster, we obtain ten solutions (Fgure 7–6).

To obtain a version of `pent_plat` , where we know at least one solution (namely the initial position of all vertices), we define ten vertices as

$$\mathbf{p}_i := \begin{bmatrix} \cos(\alpha i) \\ \sin(\alpha i) \\ e_i \end{bmatrix}, \text{ where } \alpha := 2\pi/10 \text{ and } e_i := \begin{cases} 1 & \text{if } i \text{ is odd }, \\ 0 & \text{if } i \text{ is even }. \end{cases} \qquad i = 0, 1, \ldots, 9.$$

We initialize the clusters as

$$C_1 := [\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3], \quad C_2 := [\mathbf{p}_2, \mathbf{p}_3, \mathbf{p}_4, \mathbf{p}_5], \quad C_3 := [\mathbf{p}_5, \mathbf{p}_6, \mathbf{p}_7, \mathbf{p}_8],$$
$$C_4 := [\mathbf{p}_7, \mathbf{p}_8, \mathbf{p}_9, \mathbf{p}_0], \quad C_5 := [\mathbf{p}_9, \mathbf{p}_0, \mathbf{p}_1].$$

The algorithm selects the left most minimum spanning tree (Figure 7–3), with C_5 as the home cluster. That is, the minimum spanning tree has two levels: $C_5\{C_1, C_2\}\{C_4, C_3\}$, that is C_2 is a child of C_1 and C_1 is a child of C_5 and C_3 is a child of C_4 and C_4 is a child of C_5.

The non-tree edges between C_2 and C_3, as mentioned earlier, represent a stable or independent set [13] of active overlap and distance constraints for the cluster, that force the remaining tetrahedral child clusters clusters C_1, C_2, C_3, C_4 into specific spatial arrangements:

$$\mathbf{c}_{2,4} = \mathbf{c}_{3,1} \text{ (three constraints) and } \|\mathbf{c}_{2,3} - \mathbf{c}_{3,2}\|^2 = \delta.$$

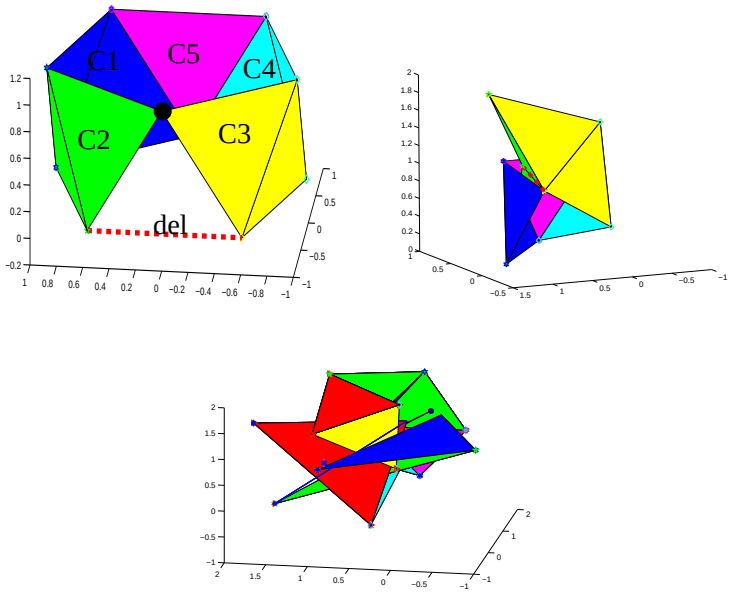

Figure 7–7: Three of the eight solutions to the `pent_plat` problem (**iv**).

iv. For $\delta = 1.381966011...$, the square of the length of the edge of a regular unit pentagon, there are eight solutions, including the initialization (Figures 7–7, 7–8).

Beyond the obvious practical usefulness, the optimization method presented here suggests that the minimum spanning tree of the cluster overlaps and the choice of home cluster (cluster C_4 for `hex_tet` and cluster C_5 for `pent_plat`) help understand the structure and symmetries of the solution space.

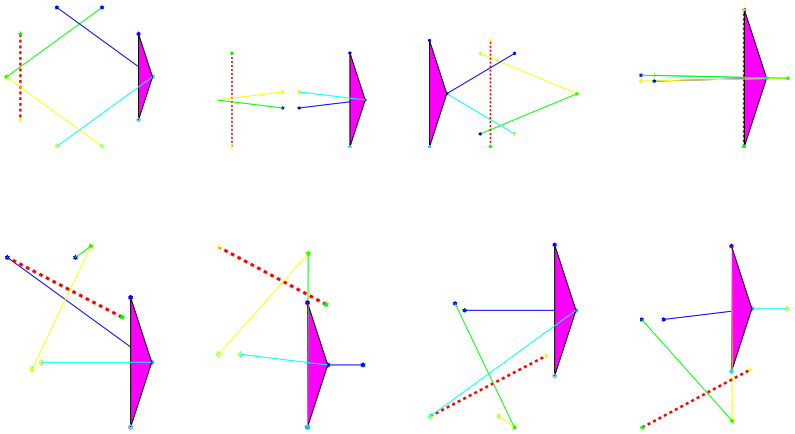

Figure 7–8: Diagramatic view of 8 solutions to the `pent_plat` problem (**iv**).

The optimization method additionally suggests a visualization of a solution path for the solved cluster as a sequence of operations on the child clusters. The unoptimized case explicitly resolves overlap constraints between clusters as part of a simultaneous system that includes active distance constraints and thereby hides the solution path.

By contrast, the optimized method uses the chosen rooted spanning tree of Section 7.2 to define the sequence of solution operations. The method first applies a sequence of unevaluated or abstract rotations to each non-home child cluster, considered in the tree order, and then translates it to its overlap points with its ancestor cluster. This leaves only the (quaternion or axis) rotations to solve for, when enforcing the distance constraints and the remaining incidence constraints.

CHAPTER 8
OPTIMAL, WELL-FORMED RECOMBINATION

8.1 Introduction

Large algebraic systems arise, for example, in industrial geometric constraint solving. Modern solvers employ sophisticated recursive decompositions to solve subsystems piecemeal and then solve *recombination* systems to combine solutions to the subsystems back into a global solution. A recombination system consists of both original constraints that relate geometric objects appearing in different subsystems and additional constraints that assert incidences between copies of shared objects that appear in different subsystems in the decomposition. The constraints between the shared objects appear in all the sharing subsystems formally lead to consistently overconstrained or dependent recombination systems. Thus the constraints between the shared objects are typically satisfied in slightly perturbed form in the sharing subsystems. Due to this perturbation, the consistency of the dependent constraints is now difficult to detect; and their solution will resist solvers, such as exact Groebner basis reduction, that use exact computations and require exact equivalence. On the other hand, due to the freedom in scaling, even small perturbations can create convergence problems for numerical solvers if the tolerances are set too tight. The recombination system can typically only be solved by generating finite precision intermediate solutions for the subsystems. These become the coefficients of the recombination system.

Since the recombination system is usually consistent, but it is hard to track the actual source of the inconsistency, we need an *algorithm to select a subset of equations* to form a well-formed recombination system that has no dependencies, but whose roots could be a small superset of the solutions. The original constraints

are then used to easily eliminate extraneous solution. Such a *well-formed recombination algorithm* that avoids numerical or algebraic treatment, but is obtained by using only the topological structure of the constraint graph, has recently been given by Sitharam [28]. However it will be seen later in Section 8.2, the resulting system is not optimized with respect to algebraic complexity.

On the other hand, for well-constrained systems, Sitharam et al. [14, 55] give an efficient graph-based algorithm to find a recombination system whose algebraic complexity is optimized. By partial elimination, this approach reduces the recombination system to a much smaller, denser one and thereby moves the problem into the range of tractability. This *optimal recombination algorithm* determines an elimination order that minimizes first the number of variables and then the degree in the recombination system. However, the algorithm is, off hand, not guaranteed to give a well-formed recombination system free of dependencies.

In the following, we combine the optimal recombination algorithm given by Sitharam et al. [14, 55] with the well-formed recombination algorithm given by Sitharam [28]. To state and prove the correctness of the new algorithm, we review the salient machinery [14, 28, 55] in Section 8.2. Section 8.3 then motivates the algorithm, shows correctness and discusses the implications.

8.2 Background: Constraint Systems and Clusters

Geometric constraint systems are succinct, conceptual and editable representations of geometric composites. They arise in applications such as mechanical computer aided design, robotics, molecular modeling and teaching geometry. For recent reviews of the extensive literature on geometric constraint solving [2, 3, 4, 6].
A *geometric constraint system* relates a finite set of primitive geometric objects (points, lines, line segments, conics, and so on) by a finite set of constraints on distance, angle, incidence, tangency. The coordinates of the objects then appear as variables of algebraic equations and inequalities. Note we will focus only on

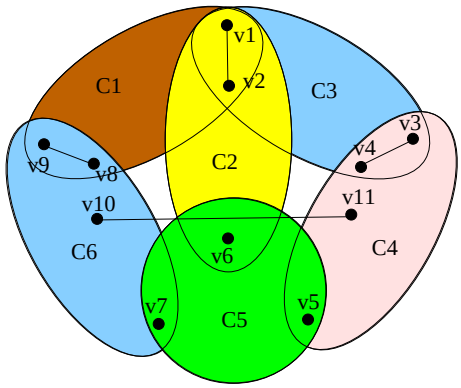

Figure 8–1: Schematic representation of a 3D geometric constraint system.

distance constraints systems where the only constraints are between points. A distance constraint of d between two points (x_1, y_1) and (x_2, y_2) in 2D is written as $(x_2 - x_1)^2 + (y_2 - y_1)^2 = d^2$, where d is the constraint parameter.

A *solution or realization* of a geometric constraint system is a *real* zero of the algebraic system. Each zero represents a positioning and orientation of the objects with respect to each other so that incidence and distance constraints hold. *Wellconstrained systems* have a finite but potentially very large number of zero-dimensional solutions. *Underconstrained systems* have infinitely many solutions and *overconstrained systems* have no solution unless they are *consistently* overconstrained. Wellconstrained or consistently overconstrained systems are called *rigid*.

For tractability, many modern geometric constraint solvers [6] represent geometric constraint systems as graphs where the vertices represent the geometric primitives and the edges present the constraints between them and employ a *decomposition-recombination (DR) plan*, a combinatorial structure that recursively decomposes an input geometric constraint system into collections of smaller, *generically* rigid, natural subsystems, called *clusters*. The clusters are recursively decomposed and solved and their solutions are recombined to yield a realization of

the input constraint system. Let G be a geometric distance constraint graph and

$$D := \{C_1, C_2, \ldots\}, \qquad C_i \subset G,$$

a collection of rigid clusters C_i of G. *Standard cluster decompositions* D have many desirable properties that help solving efficiency, detecting rigidity, incorporating feature hierarchies, dealing with under- and overconstraints, solution navigation [2].

[standard decomposition] The pair (G, D) is a *standard decomposition* of a $3D$ geometric distance constraint system if (i) The clusters in D are wellconstrained (or minimally rigid) and are maximal proper cluster subgraphs of G. (ii) The clusters in D form a complete covering set for G. (iii) For $i \neq j$, $|C_i \bigcap C_j| \leq 2$. Here (i) implies that there is no proper subgraph of G that is rigid and properly contains any of them; In (ii), *covering* set means the union of the clusters includes all the vertices and edges of G and no cluster is entirely contained in another cluster; and (iii) states that clusters do not share more than two vertices (since otherwise in 3D we could combine them into a single rigid cluster without using a recombination system where G has no over-constraints, whenever 2 clusters intersect on a pair of vertices , the vertices are connected by an edge).

Note, for the remainder of this manuscript, the only information that will be needed in a standard decomposition (G, D) is $(V_D, E_D, \{c_i\})$, where V_D and E_D denote the set of vertices and edges of G that are shared by more than 1 cluster in D. Any other information in a distance constraint graph G and the clusters in D is henceforth irrelevant. Hence we will simply identify this tuple with a standard decomposition D.

A system C (Figure 8–1) is decomposed into six clusters C_i. Recombining the realizations c_i of these clusters (themselves typically obtained by recursive decomposition and recombination) to obtain a realization c of C means choosing a coordinate system, say that of c_1 and repositioning c_2, c_3, c_4, c_5, c_6 in the

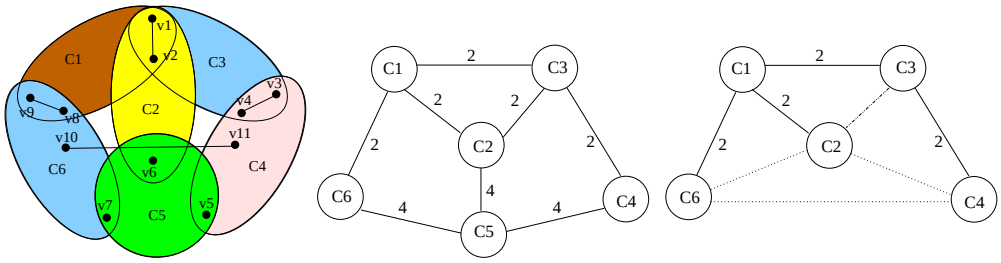

Figure 8–2: Overlap graph and the minimal spanning tree of the example (Figure 8–1).

coordinate system of c_1. Define $\mathbf{x}_{i,c_l} \in R^3$, the coordinates of v_i in c_l's local coordinate system. Given the $\mathbf{x}_{i,c_l}$, we compute translations $\mathbf{t}_{c_i} \in \mathcal{R}^3$ and the six free parameters of a 3×3 matrix $\mathbf{M}_i$ representing the composition of three rotations so that $\mathbf{x}_{i,c_1} = \mathbf{M}_i\mathbf{x}_{i,c_i} + \mathbf{t}_i$. The optimal recombination algorithm given by Sitharam et al. [14, 55] begins with an efficient representation of the transformations yields the following: For each edge shared by two clusters, one equation in one variable and for each single vertex shared by two clusters, that is not part of a shared edge, one equation in each of three variables. The number of variables is assigned as weights to an *overlap* graph (Figure 8–2) $O(H)$ of a standard decomposition $H = (V_H, E_H, \{c_j\})$: this is an undirected graph whose vertices are the clusters c_j. Its edges represent pairs of clusters.

8.2.1 The Optimal Recombination Algorithm

The optimal recombination algorithm [14, 55] takes a standard decomposition H as input and outputs another standard decomposition $D = (V_D, E_D, \{c_i\})$ by choosing an optimal covering set of clusters and further an optimal spanning tree of the overlap graph $O(D)$ of D that minimizes the sum of weights. This reduces the realization problem (Figure 8–2) from 30 equations in 30 variables to 8 equations in 8 variables.

The edge weights of the overlap graph $O(D)$ can be also viewed as follows. The degrees of freedom (dof) of rigid clusters in 3D are:

$$dof(v) = 3 \text{ for any vertex } v,$$
$$dof(e) = 5 \text{ for any edge } e,$$
$$dof(C_i) = 6 \text{ for any cluster } C_i \text{ unless } C_i \text{ is a vertex or an edge.}$$

Since a pair of vertices that have an edge between them can be moved freely but have to obey the distance constraint defined by the edge, they have 5 degrees of freedom of position and orientation. Each singleton vertex has 3 degrees of freedom. The example (Figure 8–1) has 30 independent variables. Due to the shared degrees of freedom of shared edges and shared vertices in a standard decomposition D, an edge of weight k in the overlap graph is the result of applying $6 - k$ (single coordinate) incidence constraints.

8.2.2 The Wellformed Recombination Algorithm

The *adjusted* degrees of freedom (adof) and the *removed* degrees of freedom (rdof) of any subset T of clusters in a standard decomposition D of a constraint graph G are

$$\begin{aligned} \text{adof}(G, D, T) &:= \sum_{Q \subseteq T, |Q| \geq 1} (-1)^{|Q|-1} dof(\bigcap_{C \in Q} C) \\ &=: \sum_{C \in Q} dof(C) - \text{rdof}(G, D, T). \end{aligned}$$

The key problem addressed by Sitharam [28] is that picking a well-formed recombination system in which the number of constraints matches the number of variables may contain redundant constraints and leave out essential constraints. If the solutions of the recombination system contain all the solutions of the original system, then we call it *well-formed*.

In the example (Figure 8–1), we have one distance constraint between clusters c_4 and c_6 and need to select 29 incidence constraints for recombination. Figure 8–3, *bottom*, shows two well-formed recombination systems. But the selection (Figure 8–3) has dependent incidence constraints and is hence not well-formed

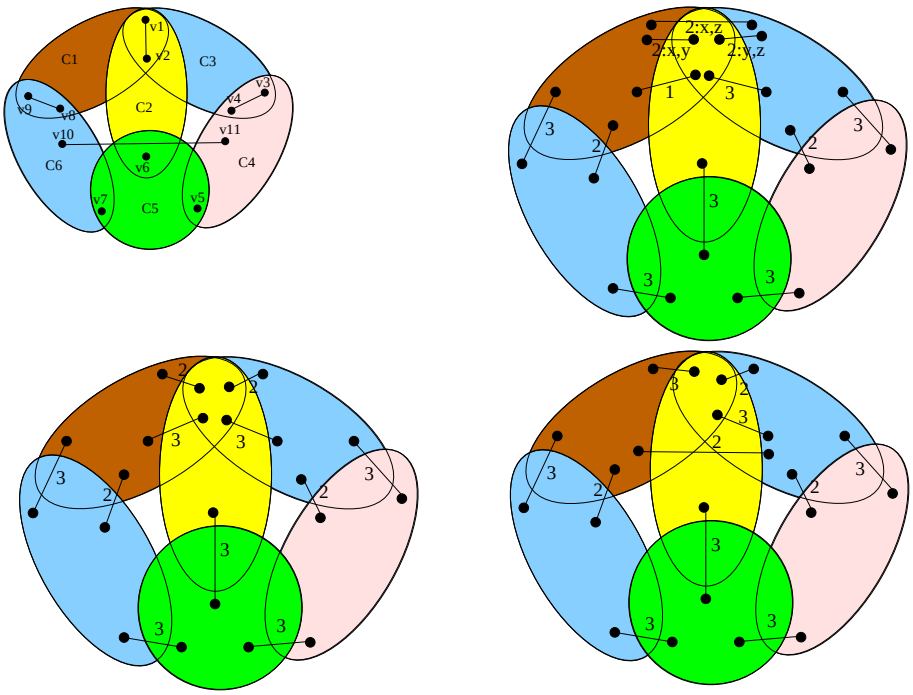

Figure 8–3: Choosing well-formed incidences for the system (Figure 8–1) is nontrivial.

For a subset of clusters $T \subseteq D$, $\mathcal{I}(\mathcal{D}, \mathcal{T})$ is the set of incidences (v, c_i, c_j, l) in $\mathcal{I}(\mathcal{D})$ for which $c_i, c_j \in T$. A cycle

$$(v, c_{i_1}, c_{i_2}, l), \ldots, (v, c_{i_{k-1}}, c_{i_k}, l), (v, c_{i_k}, c_{i_1}, l)$$

in $\mathcal{I}(\mathcal{D})$ with $k \geq 3$ is called *local cycle*.

[well-formed set of incidences] A set of incidences $\mathcal{I}(\mathcal{D})$ of a standard decomposition D of a constraint graph is *well-formed* if it (a) has no local cycle; (b) for any $T \subseteq D$, $\mathcal{I}(\mathcal{D}, \mathcal{T}) \leq rdof(D, T)$; (c) $\mathcal{I}(\mathcal{D}) = rdof(D, D)$.

Formal Problem Statement: Let $D := (V_D, E_D, \{c_i\})$ be the standard decomposition output by the optimal recombination algorithm . Give an efficient

algorithm for finding a well-formed recombination system $S(D)$ that is optimal. That is, the algorithm should minimize the number of equations in $S(D)$ and, among the minimizers, minimize the degree of the polynomials in the constraint equations.

We now describe the underlying structure of the well-formed system of incidences that is picked by the well-formed recombination algorithm. This is important to understand our contribution.

8.2.3 Seam Graphs

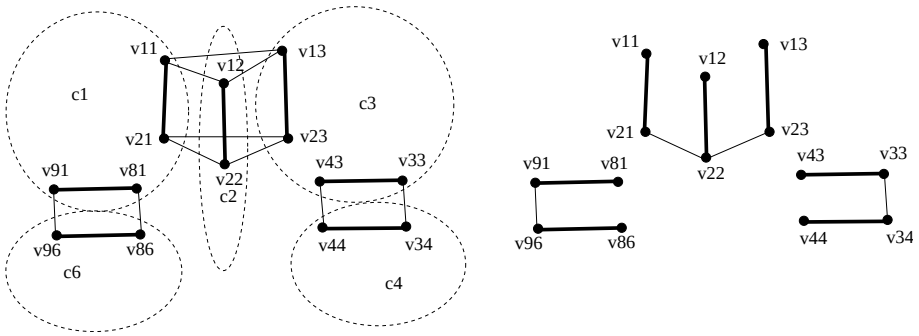

Figure 8–4: Seam graph (Figure 8–2) and its seam tree.

A *seam graph* $\mathcal{G}_D$ of the standard decomposition $D = (V_D, E_D, \{c_i\})$ of a 3D constraint graph is an undirected graph.

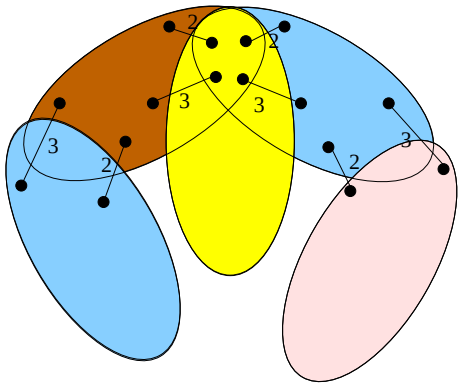

Figure 8–5: Wellformed set of incidences obtained from the seam tree (Figure 8–4).

For each $v \in V_D$, S_v is the set of c_i that contain v. The seam graph $\mathcal{G}_D$ contains $|S_v|$ copies of each vertex $v \in_D$: for each cluster c_i in S_v, we create a copy v_i of the vertex v. Such a set of vertices is denoted $\mathcal{V}_v$.

The set $mathcalE$ of $\mathcal{G}_D$ consists of two types of edges. The first is the set $\mathcal{PE}$ of *point seam* edges that connect every pair of vertices (u, w) in the set $\mathcal{V}_v$ (forming a complete graph) for each v.

For each $v \in V_D$, S_v is the set of c_i that contain v. The second is the set $\mathcal{LE}$ of *line seam* edges which consists of $|S_e|$ copies of every edge $e \in E_D$. That is, for each edge (u, w) in E_D, and each cluster c_i in S_e that shares e in G, we create a copy (u_i, w_i) of the edge (u, w). These sets of edges are denoted $\mathcal{E}_e$. Finally:

$$\begin{aligned}
\mathcal{V} &:= \bigcup_{v \in V_D} \mathcal{V}_v; \\
\mathcal{PE} &:= \bigcup_{v \in V_D} \mathcal{PE}_v = \{(u, w) : u, w \in \mathcal{V}_v\}; \\
\mathcal{LE} &:= \bigcup_{e \in E_D} \mathcal{E}_e; \\
\mathcal{E} &:= \mathcal{PE} \bigcup \mathcal{LE}; \\
\mathcal{G}_D &:= (\mathcal{V}, \mathcal{E})
\end{aligned}$$

A *seam path* is defined between or connecting a pair of vertices u, w that belong in the same set $\mathcal{V}_v$ for some vertex v of the seam graph G_D. We denote a path as a sequence of consecutively incident edges. The path assigns each of its edges a direction. A seam path between u and w, $u \neq w$, is a simple path that is the concatenation of simple path segments $h_0, g_1, \ldots, h_{2m}, g_{2m+1}, \ldots, h_{4m}$, where u and w are the first and last vertices of the first and last edge on the path; and where the h_{2j}'s could be empty and consist only of point seam edges (hence all edges in each h_{2j} necessarily are in the same set $\mathcal{PE}_v$ for some v). Each g_{2j+1} is a single edge in $\mathcal{LE}$ and has a unique partner edge g_{2l+1} such that both edges belong

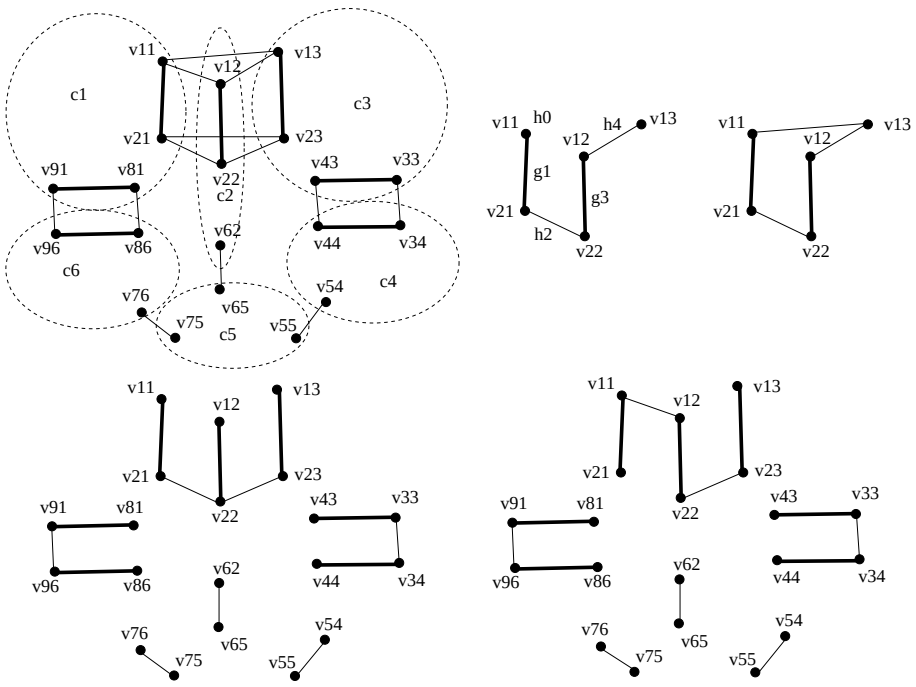

Figure 8–6: Seam graph, seam trees and seam cycle.

to the same set $\mathcal{E}_e$ for some $e = (x, y) \in E_D$. When associated with the clusters c_i and c_k in S_e, they appear directed as (x_i, yi) and (y_k, x_k) along the path. A *seam cycle* is a closed seam path, that is, a seam path between u and w, where $u = w$.

Seam subgraphs are edge induced, spanning subgraphs that is, they include all vertices, and are line inclusive, that is, they include all of the line seam edges in $\mathcal{LE}$. A *seam forest* is a seam subgraph that does not contain any seam cycles. A seam subgraph is *seam connected* if it contains a seam path between every pair of vertices that belong to the same set $\mathcal{V}_v$, for every $v \in V_D$. A *seam tree* is a seam forest that is seam connected (Figure 8–6).

The following key result from Sitharam [28] (a) shows that seam trees and local-cycle-avoiding maximal completions of them result in well-formed recombination systems and (b) gives efficient greedy algorithms exist to find them. We will use these results in Section 8.3.

Theorem 8.2.1 *Let $\mathcal{G}_D$ be the seam graph of a standard decomposition D. Take a seam forest $\mathcal{F}$ of $\mathcal{G}_D$ and let T^* be a maximal extension or completion of $\mathcal{F}$ that*

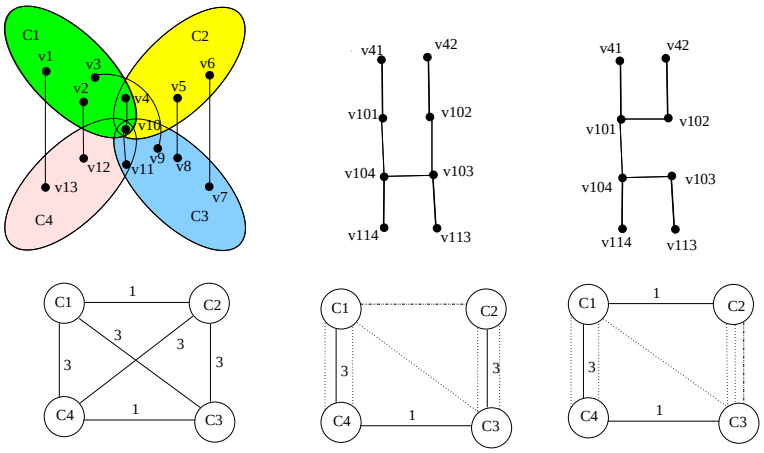

Figure 8–7: Wellformed incidences may not be optimized.

avoids local incidence cycles. Construct the set of incidences $\mathcal{I}$ *as follows. For each point seam edge* (v_i, v_j) *in* T^**, put the partial incidences* $(v, \{c_i, c_j\}, 1)$ *and* $(v, \{c_i, c_j\}, 2)$ *in* $\mathcal{I}$*. For each point seam edge* (v_i, v_j) *in* $\mathcal{F}$ *put complete incidences in* $\mathcal{I}$ *by adding on the incidence* $(v, \{c_i, c_j\}, 3)$*. If* $\mathcal{F}$ *is a seam tree,* $\mathcal{I}$ *is well-formed; If* $\mathcal{F}$ *is a seam forest and* T^* *is not necessarily a maximal extension,* $\mathcal{I}$ *satisfies the first 2 properties of well-formed incidences. There is a greedy algorithm to complete any seam forest* $\mathcal{F}$ *into a seam tree* T *containing* $\mathcal{F}$ *in time at most* $O(|G_D|) = O(|V_D + E_D||D|)$*, and a straightforward algorithm to complete any local-cycle-avoiding extension* $\mathcal{F}^*$ *of* T *into a maximal such extension* T^* *in time at most* $O(|V_D||D|)$*.*

In fact, not all wellformed sets of incidences are optimized *w.r.t.* algebraic complexity. We will show later that any spanning tree of a standard decomposition has its corresponding seam tree. Only the wellformed incidences of the corresponding seam tree of the minimum spanning tree is optimized. The corresponding wellformed incidences of the middle seam tree (Figure 8–7) is unoptimized because according to the reverse corresponding spanning tree, it needs 7 equations to solve the system. The equation set has 7 variables, 1 rotation angle variable between C_3 and C_4, 3 rotation angle variables between C_1 and C_4, 3 rotation angle variables

between C_2 and C_3. The reverse corresponding spanning tree of the right seam tree needs only 5 variables (1 rotation angle variable between C_1 and C_2, 3 rotation angle variables between C_1 and C_4, 1 rotation angle variable between C_4 and C_3).

8.3 The New Hybrid Algorithm

The optimal recombination algorithm chooses a covering set and thus a standard decomposition $D = (V_D, E_D, \{c_i\})$ and finds the minimum spanning tree ST of the overlap graph of D which is used to conduct the optimal recombination system. We now propose a method to read off the incidences system from this optimal recombination system - we will later show that this set of incidences can be greedily extended into a well-formed recombination system.

Let D be the standard decomposition and ST be the spanning tree of the overlap graph $O(D)$ output by the optimal recombination algorithm [14, 55].

Proposed system of incidences $\mathcal{J}(ST)$ corresponding to the spanning tree (ST) output by optimal recombination algorithm.

The edge with weight 3 that connects clusters c_i and c_j in ST (that is, when c_i and c_j overlap on one vertex v), yields the complete incidences $(v, \{c_i, c_j\}, 1)$, $(v, \{c_i, c_j\}, 2)$, $(v, \{c_i, c_j\}, 3)$. The edge with weight 1 that connects clusters c_i and c_j in ST (that is, when c_i and c_j overlap on one edge (u, v)), yields 5 incidences: the partial incidences of u $(u, \{c_i, c_j\}, 1)$, $(u, \{c_i, c_j\}, 2)$ and the complete incidences of v $(v, \{c_i, c_j\}, 1)$, $(v, \{c_i, c_j\}, 2)$, $(v, \{c_i, c_j\}, 3)$.

We call this system of incidences $\mathcal{J}(ST)$. We proceed to show that this system $\mathcal{J}(ST)$ is *partial well-formed system*. That is, it can be extended to a well-formed system of incidences for recombination of D. We do this in 2 steps. Theorem 8.3.2 shows that $\mathcal{J}(ST)$ satisfies properties (a) and (b) of the well-formed set of incidences. These are exactly the properties that permit extensibility into a wellformed system. Theorem 8.3.3 further gives a simple greedy algorithm to

extend $\mathcal{J}(ST)$ into a complete well-formed system. Both theorems technically use a crucial technical lemma.

To state the lemma, we need to first carefully describe a correspondence between the edges of a spanning tree of an overlap graph of a standard decomposition and the edges of the seam graph of the same standard decomposition .

Let D be the standard decomposition, $\mathcal{G}_D$ be the seam graph of D, ST be the spanning tree of the overlap graph $O(D)$ of D. The proposed complete and partial incidences define the correspondence from the edges of ST into two sets ($V(ST), U(ST)$ respectively) of point seam edges in the seam graph $\mathcal{G}_D$.

Overlap graph - seam graph edge correspondence:

The correspondence of edges of ST into $\mathcal{G}_D$: the edge with weight 3 that connects clusters c_i and c_j in ST is mapped in $\mathcal{G}_D$ to the point seam edge $(v_i, v_j) \in V(ST)$. the edge with weight 1 that connects clusters c_i and c_j in ST is mapped in $\mathcal{G}_D$ to the point seam edge $(u_i, u_j) \in U(ST)$ and the point seam edge $(v_i, v_j) \in V(ST)$. (That is, where c_i and c_j overlap on an edge (u, v).)

The reverse correspondences of edges of $\mathcal{G}_D$ in ST: condense all the vertices of $\mathcal{G}_D$ that belong to a single cluster into 1 vertex. This condensation may map 2 point seam edges (u_{ki}, u_{kj}) and (v_{ki}, v_{kj}) (at the ends of the line seam edge pair (u_{ki}, v_{ki}) and (u_{kj}, v_{kj})) of $\mathcal{G}_D$ into one edge (c_i, c_j) in ST. The point seam edge (v_{ki}, v_{kj}), that is not associated with any line seam edge pair, is mapped into the edge (c_i, c_j) in the overlap graph of D.

The edge e_2 (Figure 8–8) in the spanning tree is mapped to the point seam edge e_2 in the seam graph. The edge e_1 in the spanning tree is mapped to the line seam edges e_{11} and e_{16} and point seam edges e_{v_8} and e_{v_9} ($e_{v_8} \in V(ST)$, $e_{v_9} \in U(ST)$ or $e_{v_9} \in V(ST)$, $e_{v_8} \in U(ST)$). And for the reverse correspondences, the vertices $v_{11}, v_{21}, v_{81}, v_{91}$ of cluster c_1 in the seam graph is condensed into one vertex C_1 in the overlap graph. the vertices v_{76}, v_{86}, v_{96} of cluster c_6 are condensed

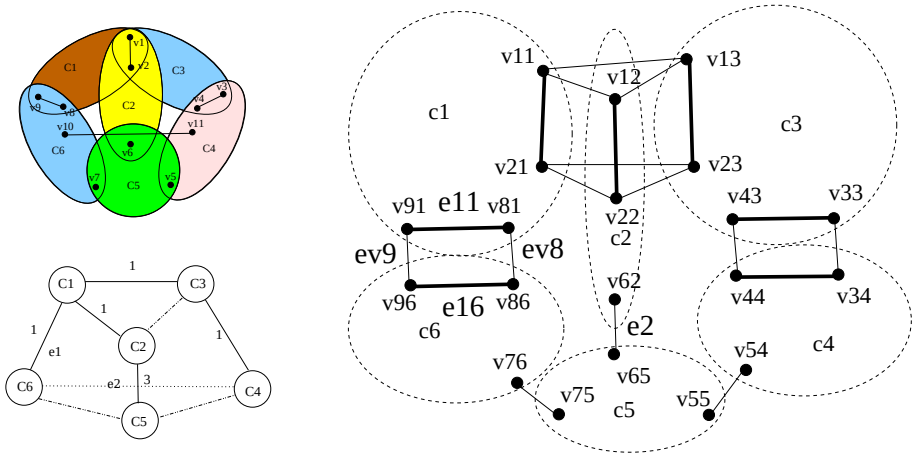

Figure 8–8: Spanning tree of its weighted overlap graph and the seam graph.

into one vertex C_6 in the overlap graph. The reverse correspondence of the two point seam edges e_{v_8} and e_{v_9} results in e_1 in the overlap graph. The vertices v_{12}, v_{22}, v_{62} of cluster c_2 are condensed into one vertex C_2 in the overlap graph. The vertices v_{55}, v_{65}, v_{75} of cluster c_5 are condensed into one vertex C_5 in the overlap graph. The reverse correspondence of the point seam edge e_2 results in e_2 in the overlap graph.

Lemma 8.3.1 *For a standard decomposition D, the seam edges in $\mathcal{G}(D)$ that correspond to the edges of in the spanning tree ST of the overlap graph $O(D)$ of D satisfy: 1) The set $V(ST)$ of seam edges that correspond to the proposed complete incidences from $\mathcal{J}(ST)$ forms a seam forest $\mathcal{F}(ST)$ in seam graph $\mathcal{G}_D$*

2) The union of $V(ST)$ and the set $U(ST)$ of seam edges that correspond to the proposed partial incidences from $\mathcal{J}(ST)$ avoids local incidence cycles

3)For any completion of $\mathcal{F}(ST)$ into a seam tree T, $\mathcal{F}^*(ST) = T \cup U(ST)$ is a local-cycle-avoiding extension of T.

Proof For the tree edge with weight 1 in ST that represents two clusters c_i and c_j overlapping on the edge (u_k, v_k), its corresponding edges in $\mathcal{G}_D$ are point seam edges allocated with 2 line seam edges (u_{ki}, v_{ki}) and (u_{kj}, v_{kj}): one of these seam edges is (u_{ki}, u_{kj}) that results in a partial incidence and the other point seam

edge (v_{ki}, v_{kj}) that results in a complete incidence. For the tree edge with weight 3 in ST that represents two clusters c_i and c_j overlapping on the vertex v_k, its correspondences in $\mathcal{G}_D$ is one point seam edge (v_{ki}, v_{kj}) that results in a complete incidence.

We show $V(ST)$ forms a seam forest, that is, it avoids seam cycles. If there is a seam cycle, let (v_{ki}, v_{kj}) be a point seam edge in that seam cycle. Then there exists a seam path with v_{ki} and v_{kj} as the end points that does not involve (v_{ki}, v_{kj}). From the definition of reverse correspondence from the seam graph to the overlap graph, there is a corresponding path between the vertices c_i and c_j in ST that does not involve the edge connecting c_i and c_j. And the reverse correspondence of (v_{ki}, v_{kj}) is an edge that connects c_i and c_j in ST. Then there is a cycle in ST contradicting its tree property.

We also show that no subset of $U(ST) \cup V(ST)$ forms a local incidence cycle. If a subset $\{(v_{k1}, v_{k2}), (v_{k2}, v_{k3}), \ldots (v_{km}, v_{k1})\}$ forms a local incidence cycle in $\mathcal{G}_D$, the reverse correspondences $(c_1, c_2), (c_2, c_3) \ldots$ and (c_m, c_1) form a cycle in ST, contradicting the tree property of the spanning tree.

From Item (1) of this Theorem and Theorem 8.2.1, we can extend $\mathcal{F}(ST)$ greedily to a seam tree T. Any extension of T with $U(ST)$ avoids local incidence cycles: If there were a local incidence cycle C, then replacing (u_{ki}, u_{kj}) in the cycle with the unique length 3 seam path connecting its end points in T, which consists of 2 line seam edges (u_{ki}, v_{ki}) and (u_{kj}, v_{kj}) and 1 point seam edge $(v_{ki}, v_{kj}) \in V(ST)$, would give a seam cycle together with the edges $C \setminus (u_{ki}, u_{kj})$. This contracts the seam tree property of T.

Theorem 8.3.2 *Given the standard decomposition D and the minimum spanning tree ST of the overlap graph of D output by optimal recombination algorithm [14, 55], the proposed system of incidences $\mathcal{J}(ST)$ satisfies the first 2 properties of well-formed system of incidences for recombining D.*

Proof From Lemma 8.3.1, we get a set of edges such that the corresponding incidences (as in Theorem 8.2.1) give a partial well-formed system of incidences satisfying the first 2 properties of the well-formed system of incidences.

Theorem 8.3.3 *Given a standard decomposition D, there is a greedy algorithm for finding an optimized well-formed set of incidences.*

Proof The algorithm is: 1. Determine the optimal spanning tree by the optimal recombination algorithm . 2. Let $\mathcal{F}(ST)$ be the seam forest formed by the set of point seam edges that correspond to the proposed complete incidences from $\mathcal{J}(ST)$. Keep adding point seam edge to $\mathcal{F}(ST)$ that does not belong in the transitive closure (of the seam paths) from $\mathcal{F}(ST)$ until no more edges can be added. This gives seam tree T. 3. Let $U(ST)$ be the set of point seam edges that correspond to the proposed partial incidences from $\mathcal{J}(ST)$. Extend $T \cup U(ST)$ greedily to a maximal local-cycle-avoiding seam graph $\mathcal{F}^*(ST)$ by adding the edges from each complete graph $\mathcal{PE}_v$ of the point seam edges associated with a vertex v of V_D. 4. For each v and each point seam edge $(v_i, v_j) \in \mathcal{P}_v \bigcap \mathcal{F}^*(ST)$, add to $\mathcal{I}(D)$ the two incidence constraints $(v, \{c_i, c_j\}, 1)$ and $(v, \{c_i, c_j\}, 2)$. 5. For each v and each point seam edge $(v_i, v_j) \in \mathcal{P}_v \bigcap \mathcal{F}(ST)$, add to $\mathcal{I}(D)$ one incidence $(v, \{c_i, c_j\}, 3)$.

The optimal recombination algorithm outputs a minimum spanning tree ST which optimizes the algebraic complexity. Using Theorem 8.3.2, we know that the corresponding system of incidences $\mathcal{J}(ST)$ satisfies the first 2 properties of well-formed system of incidences after step 1 of the algorithm. From part 3 of Lemma 8.3.1, we know that any greedy extension of $\mathcal{J}(ST)$ by completing incidences to form $T \bigcup U(ST)$ after step 3 of the algorithm is local-cycle-avoiding and hence can be extended to a maximal local-cycle-avoiding system which by Theorem 8.2.1 gives a well-formed recombination system $\mathcal{I}(D)$.

8.4 Conclusion

This chapter combines the optimal recombination algorithm [14, 55] and the wellformed recombination algorithm [28] to give an optimal well-formed recombination system for a standard decomposition.

CHAPTER 9
ENUMERATION OF PATHWAYS FOR MACROMOLECULAR

We consider the problem of explicitly enumerating and counting the assembly pathways by which an icosahedral viral shell forms from identical constituent protein monomers. This poorly understood assembly process is a remarkable example of symmetric macromolecular self-assembly occuring in nature and possesses many features that are desirable while engineering self-assembly at the nanoscale.

Sitharam and Mckenna [25, 29] give the new model of that employs a static geometric constraint graph to represent the driving (weak) forces that cause a viral shell to assemble and hold it together. The model was developed to answer focused questions about the structural properties of the most probable types of successful assembly pathways. Specifically, the model reduces the study of pathway types and their probabilities to the study of the orbits of the automorphism group of the underlying geometric constraint graph, acting on the set of pathways. The authors give a randomized algorithm to compute one measure of the probability of these pathways by faithfully sampling them.

In this chapter, we give the implementation of the algorithm and prospect to incorporate another probability measure, that will be obtained combinatorially by extending Hendricksons Theorem [30] on rigidity circuits and unique graph realization into this algorithm. This involves both the theory of geometric constraints system and implementation of the new algorithm.

9.1 Introduction

Icosahedral viral shell assembly is an outstanding example of nanoscale, macromolecular self-assembly occuring in nature [56]. Mostly identical *coat protein*

monomers assemble with high rate of efficacy into a closed icosahedral *capsid* or *shell*; onset and termination are spontaneous, and assembly is robust, rapid and economical. All of these requirements are both desirable and difficult to achieve when engineering macromolecular self-assembly.

However the viral assembly process - just like any other spontaneous macromolecular assembly process such as molecular crystal formation - is poorly understood. Answering questions about viral assembly pathways can help both to encourage macromolecular assemblies for engineering, biosensor and gene therapy applications, but and also discourage assembly for arresting the spread of viral infection.

We use the viral assembly pathway model [25, 29] that employs static geometric constraints to represent the driving (weak) forces that cause a viral shell to assemble and hold it together. The model avoids dynamics and as a result is both tractable and tunable. Preliminary predictions of this model consistently explain existing experimental observations about viral shell assembly. This model was developed to answer focused questions about the structural properties of the most probable types of successful assembly *pathways*, which are essentially directed acyclic graphs (dags) representing valid constructions (or decompositions) of the virus. The nodes of these dags are biochemically *stable* subassemblies of the complete assembly, partially ordered by containment. Specifically, the model reduces the study of pathway types and their probabilities to the study of the orbits of the automorphism group of the underlying geometric constraint graph, acting on the set of pathways. Sitharam and Mckenna [25, 29] efficient randomized algorithms are given that sample the pathway set to provide approximate answers to these questions.

Since the underlying graphs are highly symmetric polyhedral graphs, it seems a viable approach to instead explicitly enumerate these (perhaps simplified) orbits

and count their sizes. The expectation is that a hybrid of the two approaches can be developed which leverages these advantages while incorporating the full generality of the model [25, 29].

While there is a well developed structure theory of *complete* viral shells [57, 58], verified by X-ray crystallography and other experimental data, the *processes* of viral shell assembly are poorly understood. From an experimental point of view, this lack of understanding is due to the extreme speed of the assembly so that wet-lab snapshots of intermediate sub-assemblies are generally unsuccessful.

From a modeling point of view, this lack of understanding is due to the fact that prior to the recent model [25, 29], previous computational models [59, 60, 61, 62, 63, 64, 65, 66, 67] generally involve dynamics of (simplified versions) of virus assembly (further description of these approaches and comparison with the approach [25, 29], is given by Sitharam and Mckenna [25]). Dynamics were used previously even when the assembly models only sought to elucidate the structure of successful pathways.

Models whose output parameters are defined only as the end result of a dynamical process are computationally costly, often requiring oversimplifications to ensure tractability. In addition, such models are also not easily tunable or refinable since their input-to-output function is generally not analyzable and therefore do not provide a satisfactory conceptual explanation of the phenomenon being modeled.

By carefully defining the probability space, using the successful assembly as a given, the static model [25, 29] gives a method to approximately compute the probabilities of successful pathway trees/dags efficiently.

The models *input parameters* are: information extracted from (a) the geometric structure of the coat protein monomer that forms the viral shell, including all relevant (rigid) conformations, Figure 9–1; (b) the geometric and

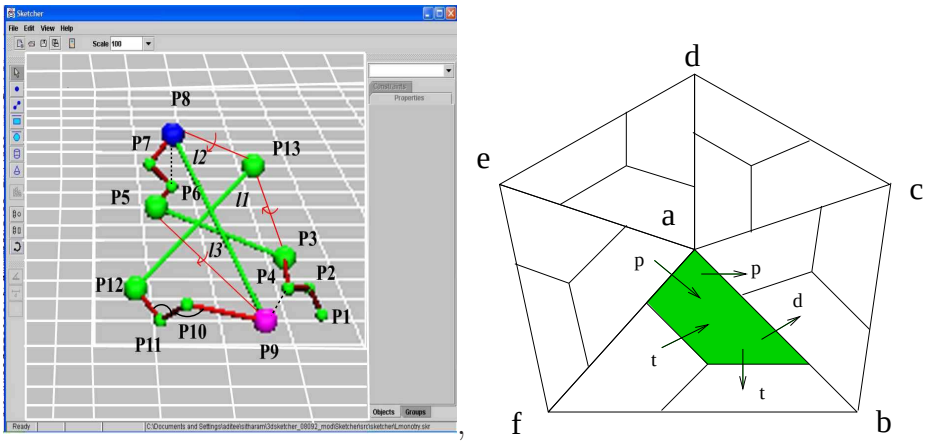

Figure 9–1: Example monomer primitives and constraints.

weak-force interactions - between pairs of monomers - that drive assembly. and (c) (optional) the inter-monomer contact or neighborhood structure of the complete viral shell, Figure 9–1.

The output information sought from the model: first, the probability that a specific *type* of successful assembly pathway incorporates a specific *type* of subassembly, leads to the complete viral shell with bounded construction *effort*; in short, a probability distribution over successful, bounded effort assembly pathways that incorporate certain substructures; this has a straightforward generalization [25] to a distribution over all possible assembly pathways (not necessarily successful) within an effort bound. The model satisfies the following requirements.

The description of the model - that is the input-to-output function - is static, that is does not rely on dynamics of the assembly process. This is achieved using the state of the art theory of 3D geometric constraint decomposition [2, 11] and is essential for forward analyzability.

The assumptions of the model are mathematically and biochemically justifiable. These justifications and rigorous comparisons of the model with existing models of viral shell assembly are given by Sitharam and Mckenna [25].

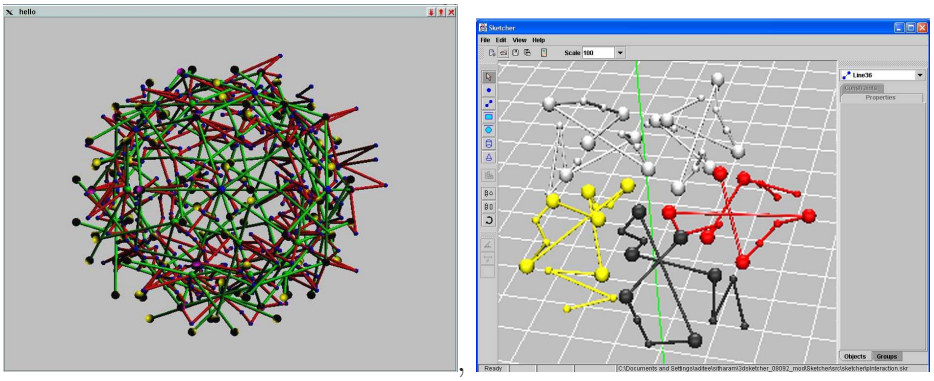

Figure 9–2: The simulated assembly of a T=1 viral shell.

The model is computationally tractable, that is *there is an efficient randomized algorithm for computing (a provably good approximation of) the pathway probability distribution.* The required algorithms are crucial modifications of state-of-the art 3d geometric constraint decomposition algorithms [2, 11]. As a result, simulation software for the model is built directly upon existing opensource software for 3D geometric constraint solving [15] (Figure 9–2).

Tractable computational simulation based on provably accurate algorithms is essential for backward analyzability which is needed for two reasons: first, for iteratively refining the model so that its output matches known biochemical information or experimental results; and second, for engineering a desired output, for example engineering the monomer structure to prevent or encourage certain subassemblies, in order to force certain pathways to become more likely than others, or to prevent successful assembly.

Preliminary simulation results [25] (Figure 9–2) show that, in principle, the model's predictions are qualitatively consistent with known studies of viruses. More conclusive biochemical validation using 3 carefully chosen, ssDNA T=1 viruses is in process [25].

9.2 Obtaining Pathway Probabilities by Combinatorial Enumeration

In this section, we first give simplified definitions of (icosahedral) viral shell graphs, valid pathways and pathway isomorphism types. For these definitions, it is viable to approach the core question of estimating specific pathway type probabilities using explicit combinatorial enumeration and counting of specific types of constructions (or decompositions) of symmetric polyhedral graphs.

These probabilities are estimated for more general definitions of pathways and more general viral shell (geometric constraint) graphs that are used in the model [25, 29], using a randomized sampling method applied to a geometric constraint decomposition algorithm. The motivation of the explicit enumeration approach is to address 2 drawbacks of this method. First, it does not utilize the icosahedral symmetry of these viruses, potentially a powerful tool in determining the pathway isomorphism classes; secondly it does not easily extend to pathways with additional properties. The expectation is that a hybrid of these two approaches can be developed which addresses these drawbacks while incorporating the full generality of the model [25, 29].

An *icosahedral* $T = m$ *viral shell graph* is obtained from a $T = m$ viral shell by representing each monomer as a vertex and each interface (relevant for assembly) between a pair of monomers by an edge.

The automorphism group of this graph is isomorphic to the icosahedral symmetry group of the viral shell. In fact, a more general result [68, 69] listing the possible automorphism groups of general polyhedral graphs could be useful for characterizing subgraphs of viral shell graphs that represent stable partial assemblies or subassemblies, whose significance will be clear below.

A *stable subgraph* S of a $T = m$ viral shell graph G is defined recursively. For the base case, a small set of at most k (independent of m) small *base stable* subgraphs of size at most $3m$ is specified and any subgraph S that is isomorphic

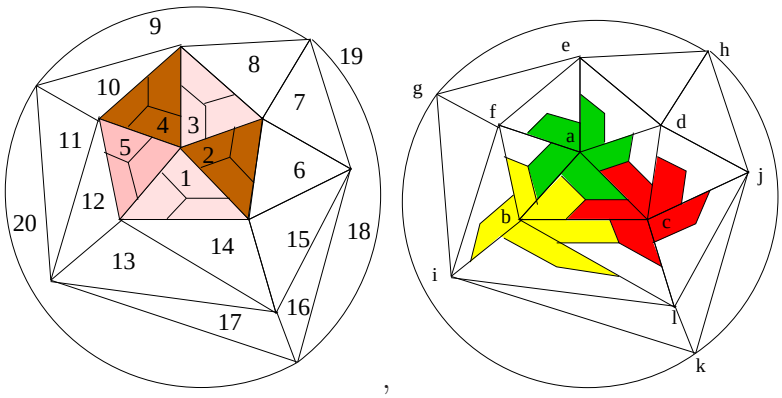

Figure 9–3: Facenumbers and vertex numbers.

to a base subgraph is stable. These constitute the *base set* B of stable subgraphs of G. For the recursion, S is stable if and only if it can be decomposed into a minimal *constituent set* Q of vertex disjoint stable subgraphs S_i such that there is a subgraph A of S that is in the base set and is not contained in the subgraph induced by any proper subset of Q.

A *stable subgraph type* is an isomorphism class obtained as the orbit of the natural action of the automorphism group of G on a stable subgraph.

For the simple T=1 viral shell graph obtained from the interfaces of Figure 9–1, the two common base stable subgraphs would be 5 cycles and 3 cycles that correspond respectively to the two common stable subassemblies, pentamers and trimers, and larger stable subgraphs would be connected subgraphs built from trimers of pentamers and pentamers of trimers respectively (Figure 9–3.

A *valid pathway* for a viral shell graph G is a tree where each node corresponds to a stable subgraph of G, the children of a parent form a constituent set for the parent, leaves are singleton vertices, and hence parents of leaves are subgraphs in the base set.

A *valid, successful pathway* is one whose root is the entire viral shell subgraph.

A *valid pathway type* is the isomorphism class obtained as the orbit of the natural action of the automorphism group of G on a valid pathway.

From the model Section 9.1 [25, 29], one of the two factors that decides the probability of a successful pathway type is the size of its isomorphism class.

9.3 Implementation Results

The expectation is that a hybrid of the explicit enumeration approach and the randomized sampling approach can be obtained which utilizes the icosahedral symmetry of these viruses and can extend to pathways with additional properties while incorporating the full generality of the model [25, 29].

To build intuition on the explicit enumeration question for the simplified pathways described above, we first implemented the randomized pathway sampling algorithm [25, 29] to sample these simplified pathways.

Implementation Result 1: For example, the implementation results for a base stable set of trimers and pentamers illuminates the structure of pathway types. For example, we obtained the highest and widest unusual pathway types (Figure 9–4).

Furthermore, using the simplified pathways described above, we believe we can begin to address the second drawback of the randomized algorithm of [25, 29] given earlier. Specifically, we are now attempting to modify the algorithm to sample only (simplified) pathways that contain a particular sub-pathway type.

Implementation Result 2: For example, the implementation results for a base stable set of trimers and pentamers shows that over 2000 trials, the largest isomorphism class had size only 6 (even we enforce the leaf subassemblies to be pentamers) (Figure 9–5). This tells us that the pathway sampling algorithm's drawbacks can indeed be debilitating: meaningful results require us to consider only pathways that contain particular subpathway types, or to consider isomorphism classes of "approximate" pathway types for some suitable definition of pathway type. For the first approach, at the moment, we are able to handle simple subpathway types: for example, we can faithfully sample pathways that are

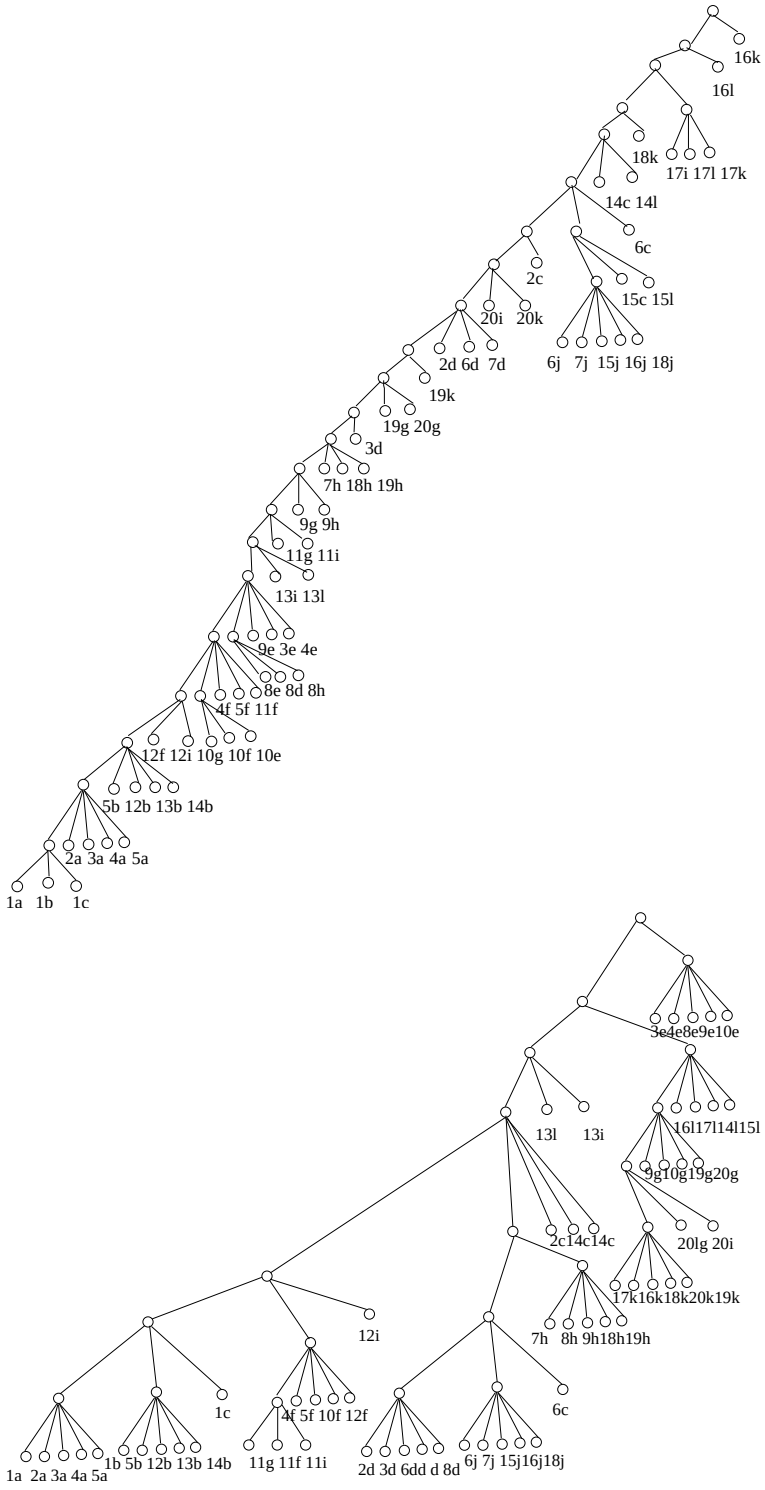

Figure 9–4: The highest pathway type and the widest pathway type in 2000 trials.

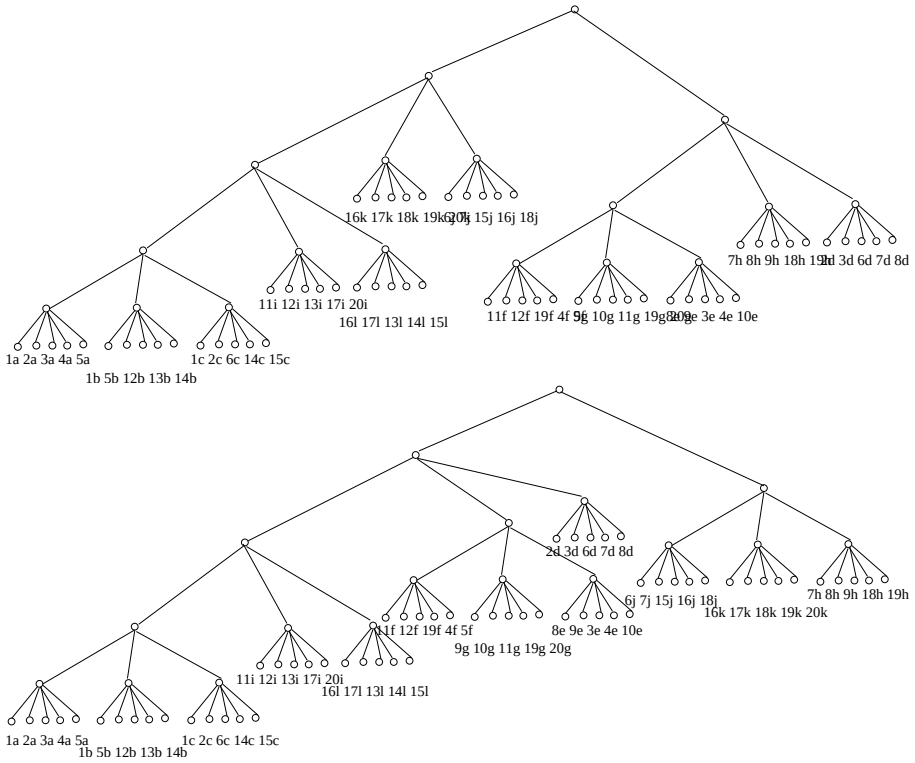

Figure 9–5: Pathways of 2 largest isomorphism classes in 2000 trials.

guaranteed to have at least one specified element of the base stable set at the bottom level (for example a trimer).

Implementation Result 3: Over 2000 trials, assigning equal probabilities for the choice of trimers and pentamers at each stage of the assembly, our results show that pathways favoring only trimers at the bottom level are significantly less likely than pathways favoring only pentamers at the bottom level. When pentamers vs. trimers were picked with probability ratio 3/5, however, the trimer favoring pathways were equally likely as the pentamer favoring pathways. It would be nontrivial to prove this type of result formally.

This brings us to the question of how the local probabilities are assigned by the randomized algorithm, that is, the probability for choosing the base stable subgraph that is used at each stage in the assembly. In our current implementation, as mentioned above, either all base stable subgraphs are equally likely to be picked

at each stage, or we assign the probability inversely to the size of the base stable subgraph, under the rationale that larger assemblies are more difficult.

Clearly the latter type of additional factor would affect the true probability of that pathway, that is, the probability distribution over pathways would incorporate an additional measure to account for the difficulty of forming particular subassemblies, in addition to just the numeracy of the isomorphism classes. For the sampling algorithm to provide a faithful sample in such a modified probability distribution over pathways, it should be modified to correspondingly to incorporate such a measure and still provide a faithful sample.

In the next subsection we propose our plans for future work on this question.

9.4 Another Probability Measure

The probability of one subassembly, corresponding to the node in the pathway, is proportional to the factor of the number of its solution types over the total number of the solution types of its subassemblies, corresponding to its child nodes in the pathway. That is, the more solution types it has and the less solution types its subassemblies have, the higher probability the subassembly has.

Due to the high dimensional solution space of the pathway, it is not tractable to count the solution types by solving it. It is desirable to have a combinatorial method to count the solution types and thereafter to get the probabilities of subassemblies. Hendrickson [30] gives a combinatorial method to get conditions for unique graph realization. We propose to extend its method to solve our problem based on the highly overconstrained property of the viral shell.

REFERENCES

[1] C. M. Hoffmann, A. Lomonosov, and M. Sitharam, "Geometric constraint decomposition," in *Geometric Constr Solving and Appl*, B. B. and R. D., Eds., 1998, pp. 170–195.

[2] M. Sitharam, "Graph based geometric constraint solving: problems, progress and directions," in *To appear in AMS-DIMACS volume on Computer Aided Design*, D. Dutta, R. Janardhan, and M. Smid, Eds., 2004.

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

[4] I. Fudos, *Constraint solving for computer aided design*, Ph.D. dissertation, Purdue University, Dept of Computer Science, 1995.

[5] C. M. Hoffmann, A. Lomonosov, and M. Sitharam, "Finding solvable subsets of constraint graphs," in *Springer LNCS 1330*, S. G., Ed., 1997, pp. 463–477.

[6] C. M. Hoffmann, A. Lomonosov, and M. Sitharam, "Decomposition of geometric constraints systems, part ii: new algorithms," *Journal of Symbolic Computation*, vol. 31, no. 4, 2001.

[7] C. M. Hoffmann, A. Lomonosov, and M. 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*, 1999, pp. 309–324.

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

[9] A. Lomonosov and M. Sitharam, "Graph algorithms for geometric constraint solving," in *submitted, based on Lomonosov's Univ Florida PhD Thesis, 04*, 2004.

[10] 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.

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

[12] M. Sitharam and Y. Zhou, "A tractable, approximate, combinatorial 3d rigidity characterization," *Fifth Automated Deduction in Geometry (ADG)*, 2004.

[13] M. Sitharam, A. Arbree, Y. Zhou, and N. Kohareswaran, "Solution management and navigation for 3d geometric constraint systems," *accepted to ACM TOG, available upon request*, 2004.

[14] M. Sitharam, J. Peters, and Y. Zhou, "Solving minimal, wellconstrained, 3d geometric constraint systems: combinatorial optimization of algebraic complexity," *Automated deduction in Geometry (ADG) 2004, available upon request*, 2004.

[15] M. 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.

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

[17] J. Owen, "www.d-cubed.co.uk/," in *D-cubed commercial geometric constraint solving software*.

[18] G. Laman, "On graphs and rigidity of plane skeletal structures," *J. Engrg. Math.*, vol. 4, pp. 331–340, 1970.

[19] J. E. Graver, B. Servatius, and H. Servatius, *Combinatorial Rigidity*, Graduate Studies in Math., AMS, 1993.

[20] H. Crapo, "Structural rigidity," *Structural Topology*, vol. 1, pp. 26–45, 1979.

[21] H. Crapo, "The tetrahedral-octahedral truss," *Structural Topology*, vol. 7, pp. 52–61, 1982.

[22] W. Whiteley, "Rigidity and scene analysis," in *Handbook of Discrete and Computational Geometry*, pp. 893 –916. CRC Press, 1997.

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

[24] A. Lomonosov, "Graph and Combinatorial Analysis for Geometric Constraint Graphs," Tech. Rep., Ph.D thesis, Univ. of Florida, Gainesville, Dept. of Computer a nd Information Science, Gainesville, FL, 32611-6120, USA, 2004.

[25] M. Sitharam and M. Agbandje-Mckenna, "Sampling virus assembly pathway: Avoiding dynamics," *accepted to Journal of Computational Biology, available upon request*, 2004.

[26] 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.

[27] M. Sitharam and Y. Zhou, "Mixing features and variational constraints in 3d," *accepted to CAD, available upon request*, 2004.

[28] M. Sitharam, "Characterizing well-formed systems of incidences for resolving collections of rigid bodies," *Submitted; available upon request*, 2005.

[29] M. Sitharam and M. Agbandje-Mckenna, "Modeling virus assembly pathways using computational algebra and geometry," in *Proceedings of the 10th Applications of Computer Algebra conference*, 2004.

[30] B. Hendrickson, "Conditions for unique graph realizations," *SIAM J. Comput.*, vol. 21, no. 1, pp. 65–84, 1992.

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

[32] 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*, vol. 8, pp. 411–440, 1995.

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

[34] R. Klein, "The role of constraints in geometric modeling," in *Geometric constraint solving and applications*, Bruderlin and R. ed.s, Eds. 1998, Springer-Verlag.

[35] J. Han and A. Requicha, "Modeler-independent feature recognition in a distributed environment," *Computer Aided Design*, vol. 30, pp. 453–463, 1998.

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

[37] J. Owen, "Algebraic solution for geometry from dimensional constraints," in *ACM Symp. Found. of Solid Modeling*, Austin, Tex, 1991, pp. 397–407.

[38] 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.

[39] C. M. Hoffmann and P. J. Vermeer, "Geometric constraint solving in R^2 and R^3," in *Computing in Euclidean Geometry*, D. Z. Du and F. Hwang, Eds. World Scientific Publishing, 1994, second edition.

[40] C. M. Hoffmann and P. J. Vermeer, "A spatial constraint problem," in *Workshop on Computational Kinematics*, France, 1995, INRIA Sophia-Antipolis.

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

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

[43] S. Ait-Aoudia, R. Jegou, and D. Michelucci, "Reduction of constraint systems," pp. 83–92, 1993.

[44] J. Pabon, "Modeling method for sorting dependencies among geometric entities," in *US States Patent 5,251,290*, Oct 1993.

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

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

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

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

[49] J. Gaukel, "Effiziente losung polynomialer und nichtpolynomialer gleichungssysteme mit hilfe von subdivisionsalgorithmen," *Ph.D. thesis*, 2003.

[50] N. Kohareswaran, "Design of a 3d graphical user interface for frontier, a geometric constraint solver graphs," *Tech.rep., Masters thesis, Univ. of Florida, Gainesville, Dept. of Computer and Information Science, Gainesville, FL, 32611-6120,USA*, 2003.

[51] A. Bjorner, M. L. Vergnas, B. Sturmfels, N. White, and G. Ziegler, *Oriented Matroids. Encyclopaedia of Mathematics. Vol. 46. Edited by G-C. Rota*, Cambridge University Press, 1993.

[52] I. Emiris and J. Canny, "A practical method for the sparse resultant," in *International Conference on Symbolic and Algebraic Computation, Proceedings of the 1993 International Symposium on Symbolic and Algebraic Computation*, 1993, pp. 183–192.

[53] B. Huber and B. Sturmfels, "A polyhedral method for solving sparse polynomial system," *Math. Comp.*, vol. 64, pp. 1541–1555, 1995.

[54] D. Eppstein, "Representing all minimum spanning trees with applications to counting and generation," Tech. Rep. 95-50, Univ. of California, Irvine, Dept. of Information & Computer Science, Irvine, CA, 92697-3425, USA, 1995.

[55] J. Peters, J. Fan, M. Sitharam, and Y. Zhou, "Elimination in generically rigid 3d geometric constraint systems," in *Proceedings of Algebraic Geometry and Geometric Modeling,Nice, 27-29 September 2004*. 2005, pp. 1–16, Springer Verlag.

[56] D. B. S. L. G. I. W. Goddard, *Handbook of nanoscience engineering and technology*, CRC press, 2002.

[57] F. Crick and J. Watson, "Structure of small viruses," *Nature*, vol. 177, pp. 473–475, 1956.

[58] D. Caspar and A. Klug, "Physical principles in the construction of regular viruses," *Cold Spring Harbor Symp Quant Biol*, vol. 27, pp. 1–24, 1962.

[59] B. Berger, P. Shor, J. King, D. Muir, R. Schwartz, and L. Tucker-Kellogg, "Local rule-based theory of virus shell assembly," *Proc. Natl. Acad. Sci. USA*, vol. 91, pp. 7732–7736, 1994.

[60] B. Berger and P. Shor, "Local rules switching mechanism for viral shell geometry," *Technical report, MIT-LCS-TM-527*, 1995.

[61] A. Zlotnick, J. Johnson, P. Wingfield, S. Stahl, and D. Endres, "A theoretical model successfully identifies features of hepatitis b virus capsid assembly," *Biochemistry*, vol. 38, pp. 14644–14652, 1999.

[62] A. Zlotnick, R. Aldrich, J. M. Johnson, P. Ceres, and M. J. Young, "Mechanisms of capsid assembly for an icosahedral plant virus," *Virology*, vol. 277, pp. 450–456, 2000.

[63] A. Zlotnick, "To build a virus capsid: an equilibrium model of the self assembly of polyhedral protein complexes," *J. Mol. Biol.*, vol. 241, pp. 59–67, 1994.

[64] D. Rapaport, J. Johnson, and J. Skolnick, "Supramolecular self-assembly: molecular dynamics modeling of polyhedral shell formation," *Comp Physics Comm*, 1998.

[65] J. E. Johnson and J. A. Speir, "Quasi-equivalent viruses: a paradigm for protein assemblies," *J. Mol. Biol.*, vol. 269, pp. 665–675, 1997.

[66] V. S. Reddy, H. A. Giesing, R. T. Morton, A. Kumar, C. B. Post, C. L. Brooks, and J. E. Johnson, "Energetics of quasiequivalence: computational analysis of protein-protein interactions in icosahedral viruses," *Biophys*, vol. 74, pp. 546–558, 1998.

[67] C. J. Marzec and L. A. Day, "Pattern formation in icosahedral virus capsids: the papova viruses and nudaurelia capensis β virus," *Biophys*, vol. 65, pp. 2559–2577, 1993.

[68] L. Babai and W. Imrich, "On groups of polyhedral graphs," *Discrete Math.*, vol. 5, pp. 101–103, 1973.

[69] L. Babai and W. Imrich, "Sense preserving groups of polyhedral graphs," *Monatsh. Math.*, vol. 79, pp. 1–2, 1975.

BIOGRAPHICAL SKETCH

Yong Zhou was born in 1976 in Hubei, China. He grew up mostly in Anlu, Hubei. He earned his B.S. in Information and Control and his M.S. in Electrical Theory from the Shanghai Jiaotong University (SJTU) in 1999 and 2002, respectively. He earned his Ph.D. in Computer Science from the Univecity of Florida (UFL) in 2006.