DIMACS Series in Discrete Mathematics
and Theoretical Computer Science

Combinatorial approaches to Geometric Constraint Solving: Problems, Progress and Directions

Meera Sitharam

Abstract. We formulate the fundamental combinatorial problems in geometric constraint solving that have emerged in the past 15 years within the mechanical CAD context. We discuss recent progress on these problems and identify future directions along with promising techniques from related areas.

Organization

1. Background

Geometric constraint systems have been studied in the context of variational constraint solving in CAD for nearly 2 decades [**VLL81, Nel85, Tod89, Rol91, Kra92, LM96, MR97, SAK93**] [**BFH+95, Fud95, HV95, HJa97, HLS97a**] [**HLS98a, HLS01a, HLS01b, HLS99**] [**HY00, HC01, HLS99, GC98a, GC98b, Owe91**] [**Owe93, Bru86, Owe, Pab93, AJM93**] [**HSY04, LS04, OSMA01,**

Key words and phrases. Geometric constraint solving, Algorithms, Rigidity theory, Geometric modeling, Mechanical computer aided design.

The author was supported in part by NSF Grants EIA 0096104 and CCR 9902025.

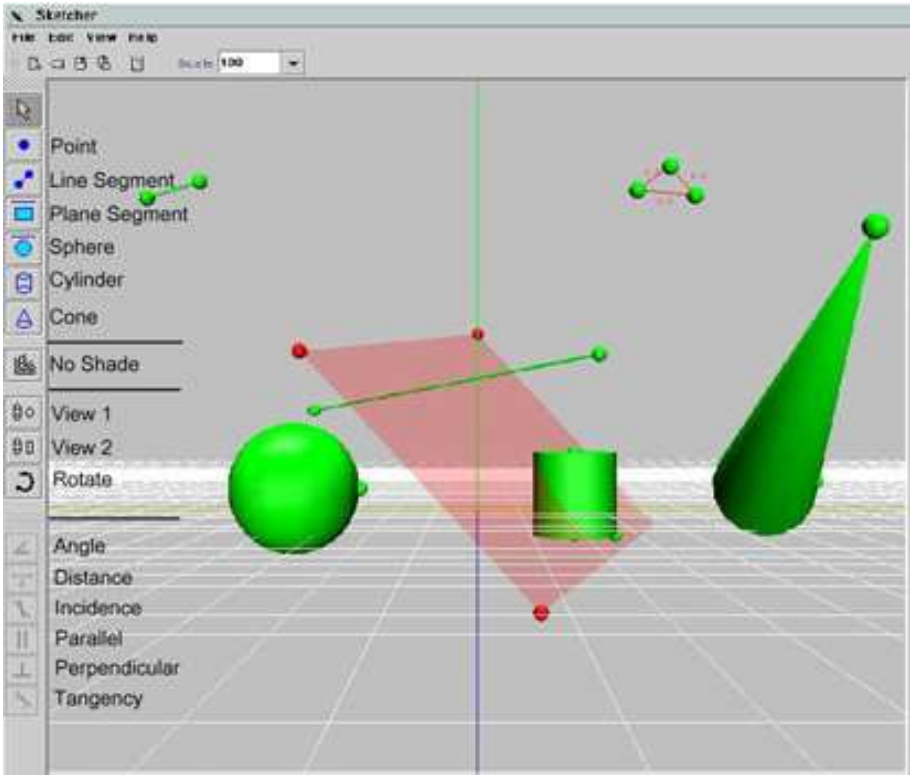

FIGURE 1. Typical 3D sketcher: icons show repertoire of object and constraint typ es

Sit04b]. For recent reviews of the extensive literature on geometric constraint solving see, e.g, [**HLS98a, Kra92, Fud95**].

A *geometric constraint system* consists of a finite set of geometric objects and a finite set of constraints between them. See Figure 1. The constraints can usually be written as algebraic equations and inequalities w hose 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 $(x_2 - x_1)^2 + (y_2 - y_1)^2 = d^2$ Most of the constraint solvers so far deal with 2D constraint systems, although some of the newer approaches including [**HLS97b, HLS98b, HLS01a, HLS01b**] [**HLS99, Bru86, Owe**] [**HSY04, LS04, OSMA01, Sit04b**], extend to 3D constraint systems. 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 an d any other parameters of) the geometric elements such that all constraints are sa tisfied. 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*, *well-constrained*, or *underconstrained*. Well-constrained systems have a finite, albeit potentially very large number of *rigid* solutions; th eir solution space is a zero-dimensional variety. 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*. Well or overconstrained systems are called *rigid* systems.

1.0.1. *Scope.* Geometric constraints have been used as succinct, minimal representations of geometric composites in many other applications besides CAD, such as robotics, molecular modeling and teaching geometry. As a result, there is a vast body of literature related to geometric constraint solving. This manuscript's scope is restricted to combinatorial methods for geometric constraint solving, specifically within the CAD application context.

Technically, geometric constraint solving straddles combinatorics, algebra and geometry. Specifically, it overlaeps combinatorial rigidity theory, geometric methods for kinematics, robotics and mechanisms, automated geometry theorem proving, geometric invariant theory, and computational algebraic geometry (solving specific classes of polynomial systems arising from geometric constraints).

The question: *to what extent can geometric constraint problems be approached combinatorially?* is highly non-trivial and important. It provides a framework for a systematic discussion of progress in the area and leads to a natural organization of the problems to be discussed in this paper.

1.1. Constraint Graphs and Degrees of Freedom. 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 2, 3, 4 show 2D and 3D constraint systems and their respective dof constraint graphs. Several more 3D constraint systems whose graphs have vertices of weight 3, 5 and edges of weight 1,3 can be found in Figures 9, 8, 5, 6, 7.

Note that the constraint graph could be a *hypergraph*, each hyperedge involving any number of vertices. A subgraph $A \subseteq G$ that satisfies

$$\sum_{e \in A} w(e) + D \geq \sum_{v \in A} w(v) \tag{1.1}$$

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 $D = 0$.

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

A dense graph with density strictly greater than $-D$ is called *overconstrained*. A graph that is dense and all of whose subgraphs (including itself) have density at most $-D$ is called *wellconstrained*. A graph G is called *well-overconstrained* if it satisfies the following: G is dense, G has atleast one overconstrained subgraph, and has the property that on replacing all overconstrained subgraphs by wellconstrained

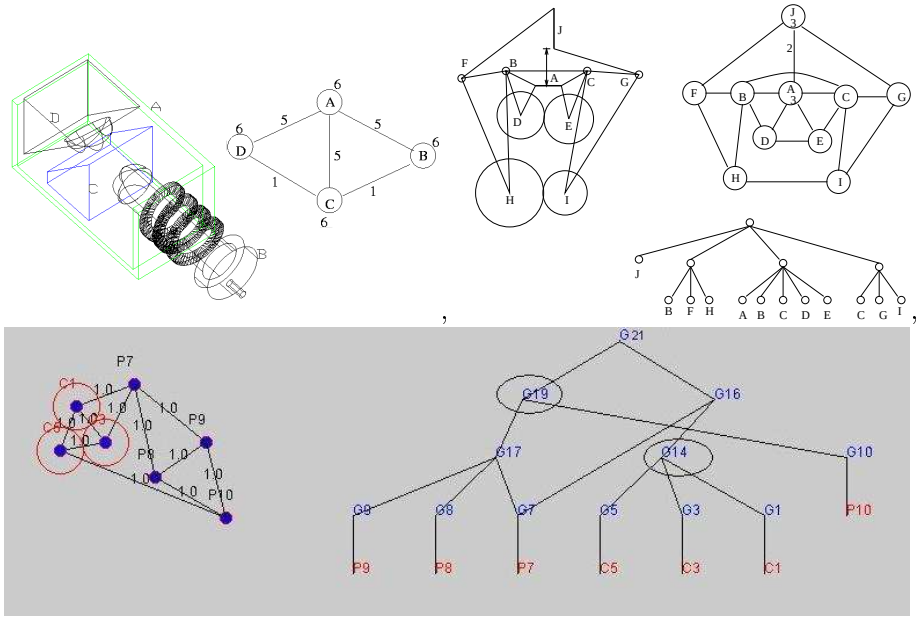

FIGURE 2. *Left:* An underconstrained 3D example with 1 extra dof and its dof constraint graph: each of 4 rigid objects has 6 dofs, B, C, D are constrained by incidence with the housing A, which removes 5 dofs in each case, so that each of the pairs AB, AC, AD is underconstrained and has one extra degree of freedom. The remaining 2 pairs BC and CD are connected by distance constraints, which remove 1 dof each. *Right:* A 2D constraint system with points or fixed radius circles and distance constraints, except for A and J which are line segments with a perpendicularity and distance constraint between them. In the constraint graph, all unnumbered vertices have weight 2 and unnumbered edges have weight 1. *Below:* A 2D example, constraint graph, DR-plan.

subgraphs (in any manner), G remains dense. A graph that is wellconstrained or well-overconstrained is called *cluster*. A dense graph is *minimal* if it has no dense proper subgraph. A graph that is not a cluster is said to be *underconstrained*. If a dense graph is not minimal, it could in fact be an underconstrained graph: the density of the graph could be the result of embedding a subgraph of density greater than $-D$.

1.2. Inadequacy of a pure dof analysis. Note that all minimal dense subgraphs are clusters but the converse is not the case.

To illustrate the challenge involved, we need to discuss how the graph theoretic properties described above based on *degree of freedom (dof) analysis* relate to corresponding properties of the corresponding constraint *system*. For this, we need to introduce the notion of *genericity*. Informally, a constraint system is generically *rigid* if it is rigid (does not flex or has only finitely many non-congruent, isolated solutions) for most of choices of coefficients of the system. More formally we use the notion of genericity of e.g, [**CLO98**]. A property is said to hold *generically* for

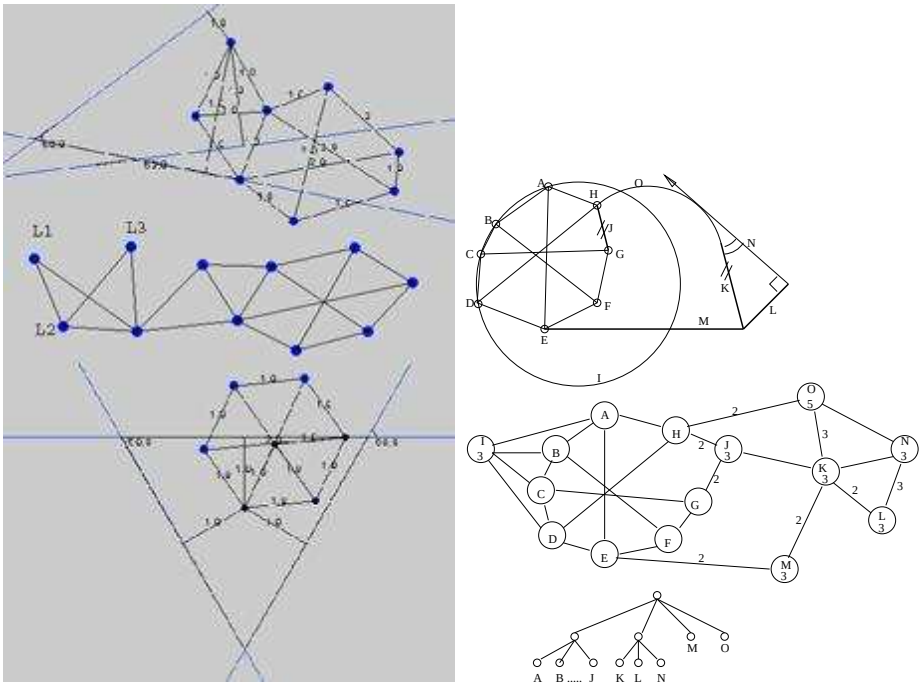

FIGURE 3. Two more 2D constraint systems examples; constraint graphs; DR-plans; all unnumbered vertices and edges have dof weights 2 and 1. *Left:* vertices L1, L2, L3 in constraint graph represent the line objects; solution shown at bottom. *Right:* Example of Variable radius Circle, Arc and Ray objects, Angle, Paralellism and Tangency constraints

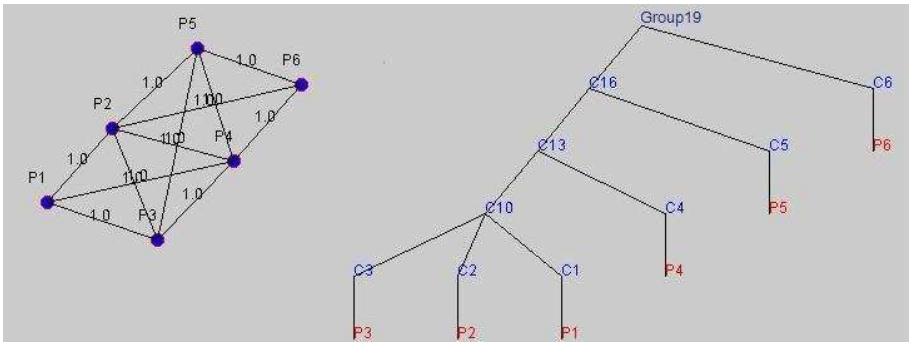

FIGURE 4. Simple 3D constraint system drawn on 2D canvas with points and distances and DR-plan

polynomials $f_1, \ldots, f_n$ if there is a nonzero polynomial P in the coefficients of the f_i such that this property holds for all $f_1, \ldots, f_n$ for which P does not vanish.

Thus the constraint system E is generically rigid if there is a nonzero polynomial P in the coefficients of the equations of E - or the parameters of the constraint system - such that E is solvable when P does not vanish. For example, if E

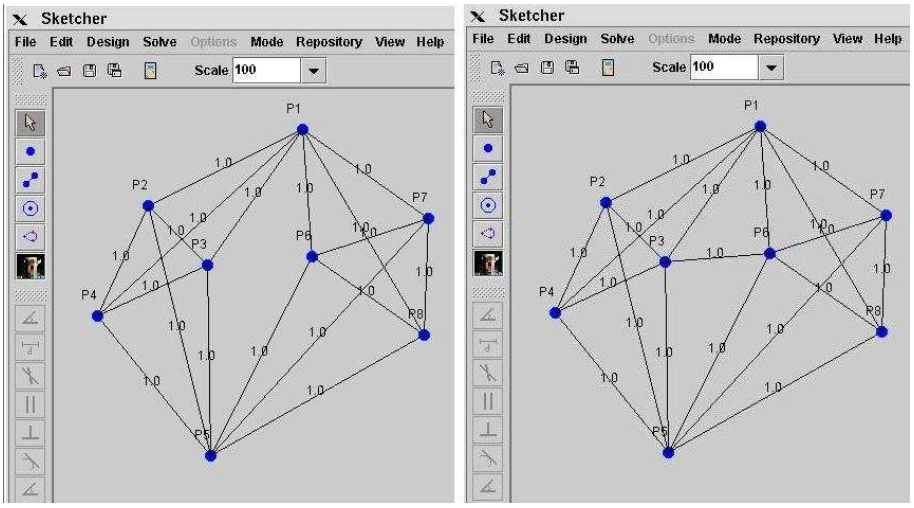

FIGURE 5. Constraint dependences in 3D: constraint system drawn on a 2D canvas; corresponding constraint graphs have vertices of weight 3 and edges of weight 1

consists of distance constraints, the parameters are the distances. Even if E has no overt parameters, i.e, if E is made up of constraints such as incidences or tangencies or perpendicularity or parallelism, E in fact has hidden parameters capturing the extent of incidence, tangency, etc., which we consider to be the parameters of E.

A generically rigid system always gives a cluster, but the converse is not always the case. In fact, there are well-constrained, dense clusters whose corresponding systems are not generically rigid and are in fact generically not rigid due to the presence of generic *constraint dependences*. Consider for example Figures 5, 6, 7 that illustrate the "bananas" problem of [**GSS93**] caused by generic constraint dependence. This problem is also present in Figure 9, and can be detected as the root cause beneath large class of combinatorial misclassifications, although this detection is nontrivial.

A combinatorial dof analysis of the 3D constraint system in Figure 6(top) would correctly report the left and right subsystems ($P1, P2, P3, P4, P5$ and P_1, P_6, P_7, P_8, P_5 respectively) and the whole system to be wellconstrained clusters. Figure 7 (bottom) has the same number of constraints, but a dof analysis

would correctly report both that the left subgraph as overconstrained and the whole as underconstrained. Figure 5 (left) also has the same number of constraints and appears to be a wellconstrained cluster according to a dof analysis. However, while the left and right subsystems are (in fact) wellconstrained clust ers, the whole system is generically overconstrained. In a well-defined sense, this is an overconstraint that is not *explicit*, but is caused by a constraint *dependence* that is not detectable by a dof or density count. However, when *restricted to consistently overconstrained situations* (those choices of distances - such as in this example - that are guaranteed to admit a solution), the system in Figure 5 (left) is generically underconstrained, although the system on Figure 5(right) is generically wellconstrained.

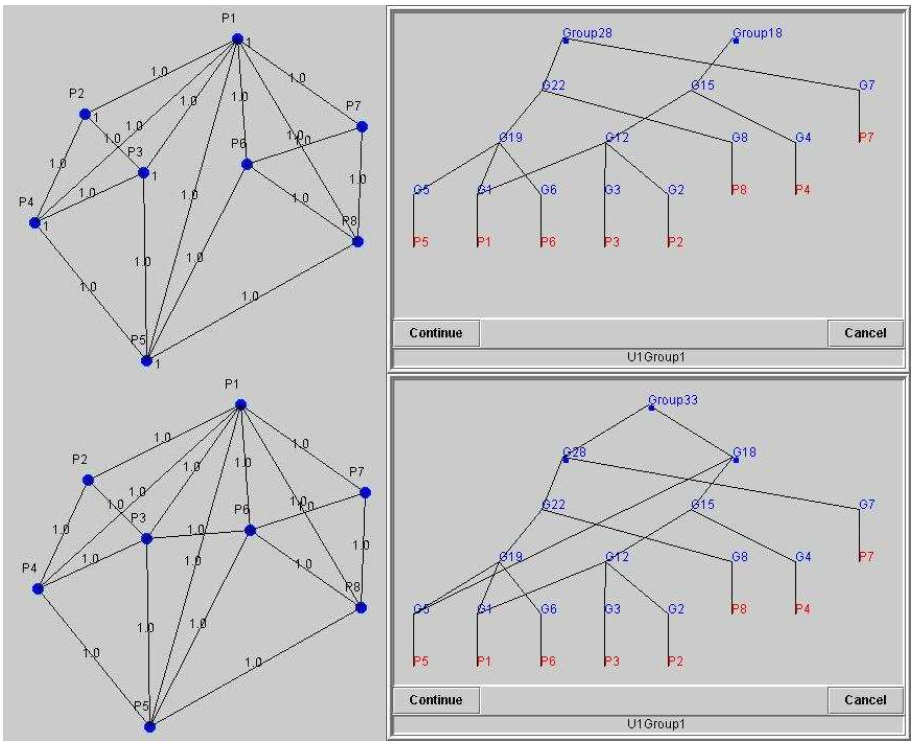

FIGURE 6. Modifications to 3D system in Figure 5: Combinatorially well con strained (DR-plan has single source, top) and underconstrained (DR-plan has many sources, bottom)

In fact, a constraint system has a constraint dependence if the common overlap of *any* subset of its wellconstrained clusters is underconstrained. The above "bananas" is a special case of this. However, among these situations, the dof analysis is inaccurate only in the "bananas" case. To date, there is no known, tractable characterization of generic rigidity of systems for 3 or higher dimensions, based purely on combinatorial properties of the constraint graph [**Whi97**], [**GSS93**], although several conjectures exist.

However, it should be noted that in 2 dimensions, according to Laman's theorem [**Lam70**], 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. But there are no known combinatorial characterizations of 2D rigidity, when other constraints besides distances are involved. For methods that address 2D systems with angle and incidence constraints, see Section 8 and [**JJO04**].

Note. In this manuscript, we will nevertheless rely on the above dof analysis, augmented, however, to check for many types of constraint dependence (including the bananas situation and others such as the so-called "hinge" situation [**Cra79, Cra82**]). Since a complete description of this augmented property of a cluster (called *module-rigidity*) [**SZ04b**] is involved and outside the scope of this paper, from now on we will use the terms *rigid system* and the definition of *cluster*

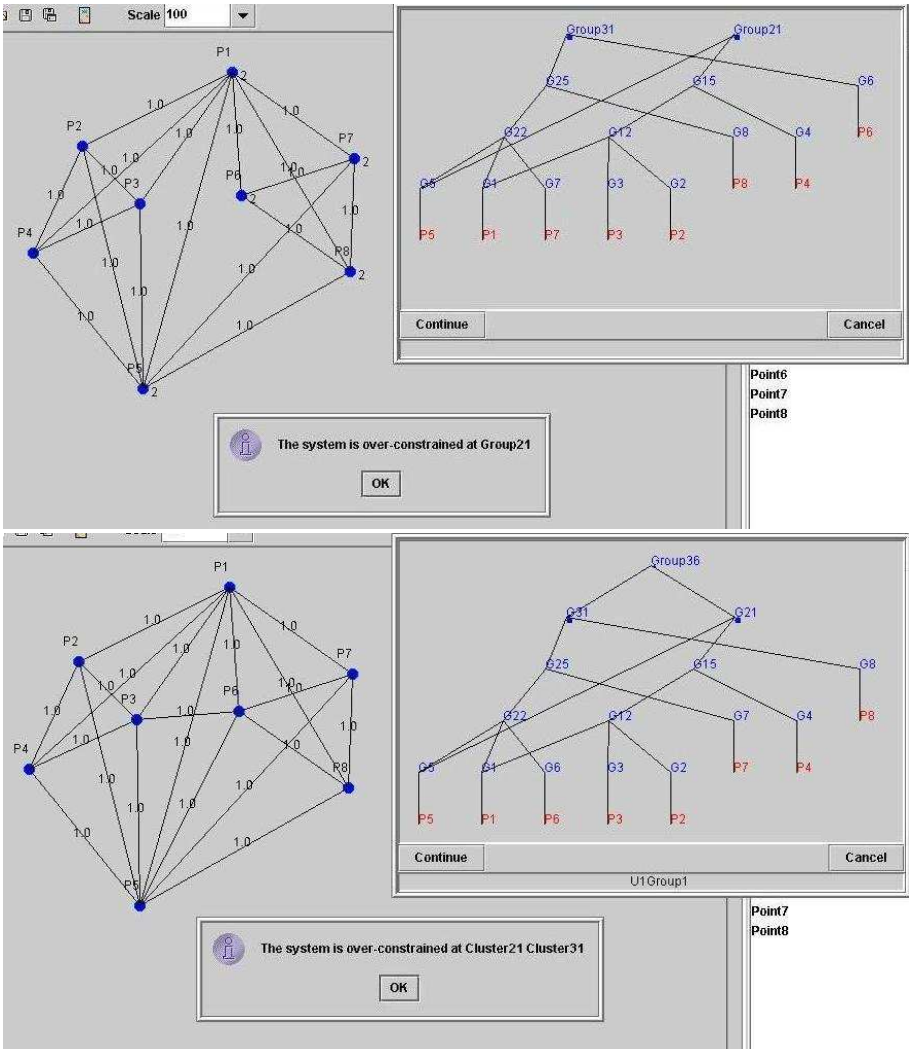

FIGURE 7. Combinatorially or explicitly overconstrained clusters in well (single source in DR-plan) and underconstrained (multiple sources in DR-plan) graphs

interchangeably. In Section 4 however, we will sketch an intuitive, simple algorithm that approximates a check for module-rigidity.

2. Basic Input-Output requirements of a geometric constraint solver

The generic input to an effective geometric constraint solver for CAD applications consists of the following.
(1) A 2D or 3D geometric constraint system (see Section 1), S, i.e, a set of primitive geometric objects, each of a specified type, and constraint types relating pairs (or larger subsets) of these objects. See Figure 1. Some constraint types have a metric value specification, for example, a distance constraint between 2 points and a value for the distance; or a torsion angle constraint between a pair of line segments in

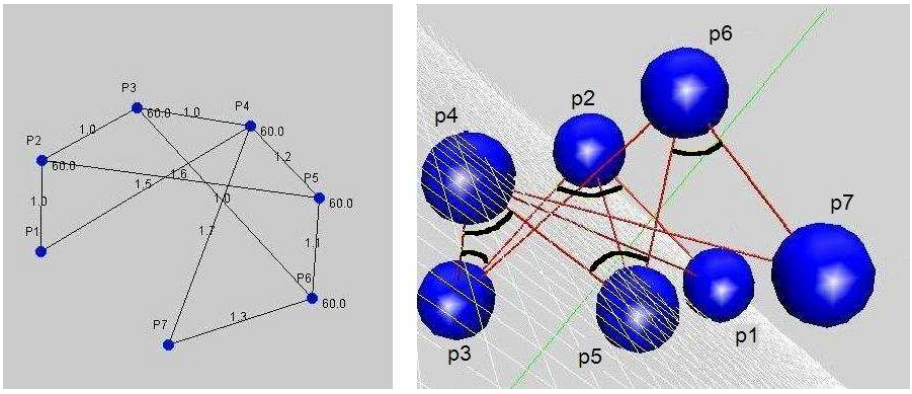

FIGURE 8. 3D constraint system drawn on 2D canvas with line and point objects and distance and angle constraints (graph has vertices of weight 3(points and lines); edges of weight 3 (incidences) and 1 (distances and angles). Solution (right)

3D and a value for the angle. The values associated with the constraints could be constants or variables related to other such values in the case of so-called *relational or engineering constraints or navigation constraints*, see Section 5. In addition, *intervals* of values are permitted for some of the incidence or intersection constraints which are effectively chirality or oriented matroid constraints, declaring (non) intersections of convex hulls of subsets of geometric objects, and used primarily for navigation of the solution space, for example, the constraint that 2 line segments should or should not interesect. The constraint system, also in 3D, is often sketched on a 2D canvas. See Figures 9 and 8.
(2) An input partial decomposition of the constraint system P obtained from one or more views of a feature, part or subassembly design hierarchy. The features or subassemblies are nodes of directed acyclic graph (*dag*) and are intended to be manipulated independently within their own local coordinate systems, as well as moved around relative to each other in the coordinate system of a higher level feature or subassembly. There could be a *priority order* given to the resolution of the features or subassemblies: a linear order degenerates to so-called parametric constraint solving. See Figure 10.
(3) Some of the features or subassemblies in the input hierarchy may have been already solved and may be chosen from a repertoire of commonly occuring such features, parts or subassemblies. In this case, they may not be input as constraint systems; they should be flagged and treated en-bloc as already solved rigid bodies.
(4) Additions, changes, updates to all of the above. See Figure 28.
(5) Other interactive user input requested by the geometric constraint solver during solution space navigation.

A geometric constraint solver's desired outputs are the following.
(1) *Classification* of the input constraint system S as (generically) wellconstrained, overconstrained or underconstrained. Classification of each feature or subassembly in the hierarchy P as wellconstrained, overconstrained or underconstrained. See Figures 6, 7, 5.

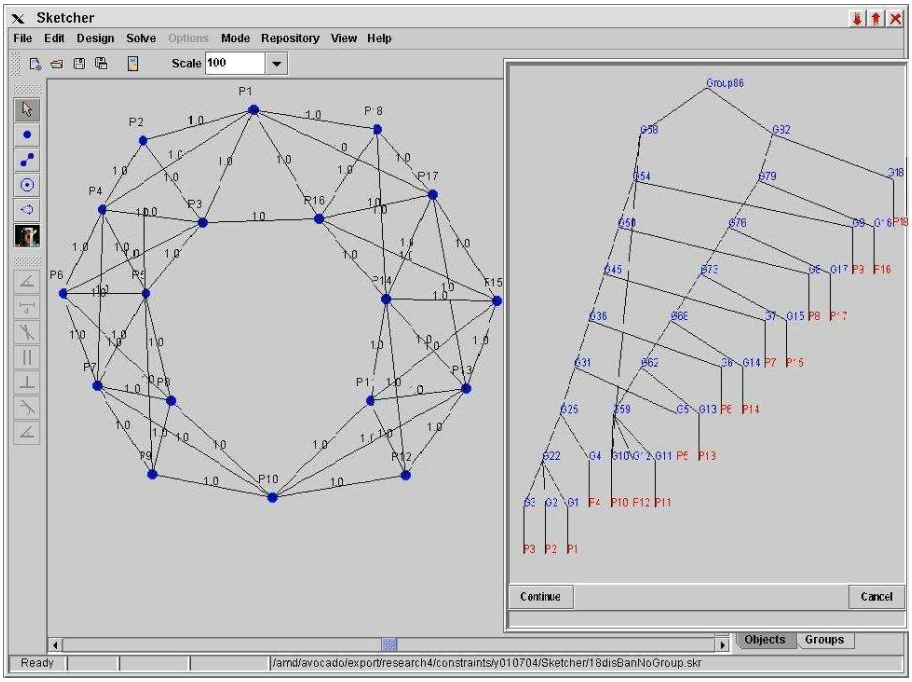

FIGURE 9. 3D constraint system (with generic constraint dependence not detectable by a simple dof count), sketch on a 2D canvas; it is wellconstrained for the given set of distances; corresponding DR-plan

(2) In the case the system is classified as wellconstrained, a method for systematically *navigating* the solution space with or without user intervention, specifically, steering through the various solutions, realizations or conformations of each of the wellconstrained features, parts and subassemblies of the associated input partial decomposition P, in priority order (if one is given) See Figures 20, 21, 22, 23, 24.
(3) In case combinatorially overconstrained, potentially inconsistent, features, parts, subassemblies or subsystems are present, see Figures 7, for any requested collection of them, give a complete, navigable representation of the wellconstrained *reductions*; for example an exhaustive list of existing constraints any one of which can be removed without generically making the entire system or any of the subsystems - i in the collection - underconstrained. For each such removal, a second exhaustive list of constraints that can be removed, and so on, resulting in a sequence of lists.
(4) In case of those (sub)systems classified as underconstrained (*ambiguous*), a *complete* decomposition into maximal clusters (i.e., where all the maximal clusters are present). In addition, a complete, tractable, navigable representation of its wellconstrained *completions*. This question again has a well-defined interpretation using rigidity matroids which we omit here. One such navigable representation could be an exhaustive list of constraints (participating objects and type of constraint, but not value) that can be added without generically making the system overconstrained. For each such addition, a second exhaustive list of constraints that can be added, and so on, resulting in a sequence of lists.

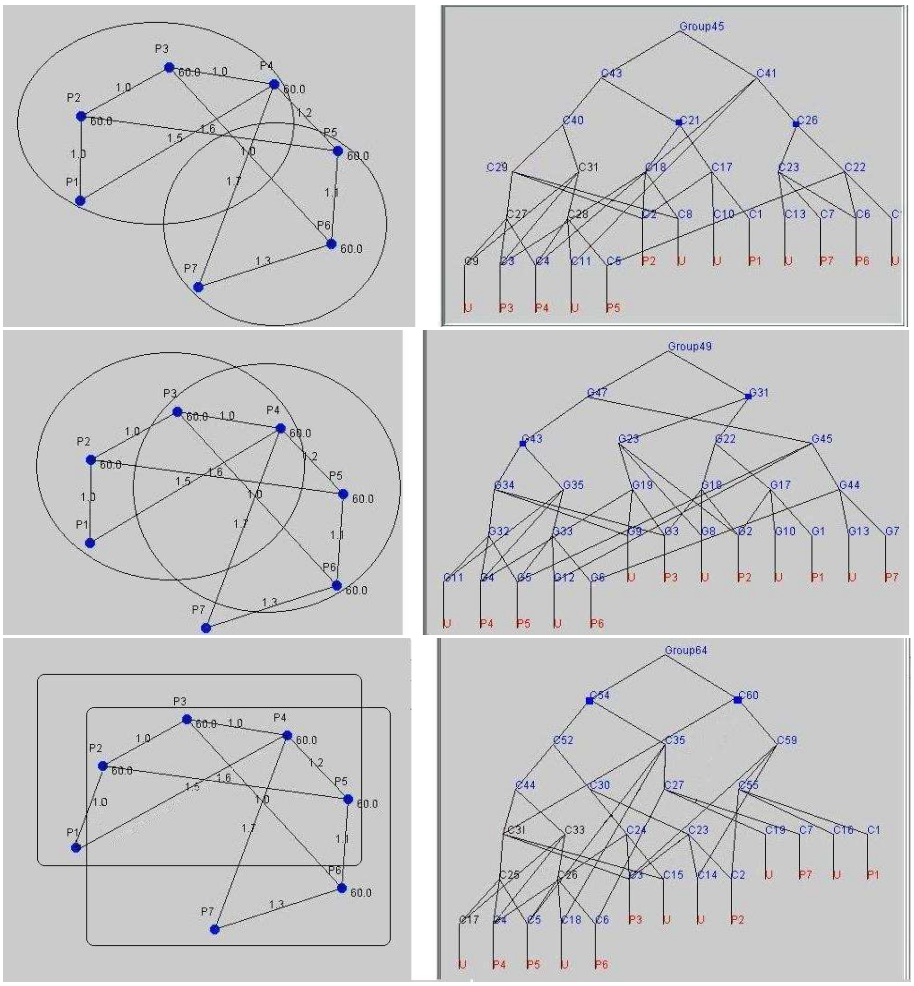

FIGURE 10. Left: input partial decompositions for 3D constraint system shown in Figure 8; groups are features or subassemblies. Right: 3 different DR plans incorporating corresponding input decomposition. Features appear as clusters or, if underconstrained, their complete set of maximal clusters. Features may (not) intersect on (non) trivial subgraphs.

(5) A method of efficiently *updating* the output when incremental changes are made to the input. For example: a constraint value is changed, a constraint is added or removed, an object is added or removed, a feature is added etc. These could be offline or online updates (see settings of a typical constraint solver below).

2.1. The need for decomposition: DR-plans and their properties. The overwhelming cost of solving a geometric constraint system is the size of the largest subsystem that is solved using a direct algebraic/numeric solver. This size

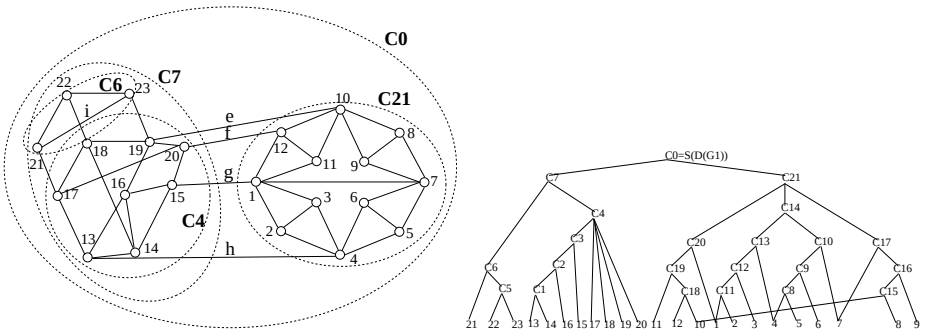

FIGURE 11. 2D constraint graph G1 and DR-plan; all vertices have weight 2 and edges weight 1

dictates the practical utility of the overall constraint solver, since the time complexity of the constraint solver is at least *exponential* in the size of the largest such subsystem.

Therefore, an effective constraint solver should *combinatorially* develop 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 are generically rigid (have only finitely many solutions). The DR-planner is a graph algorithm that outputs a *decomposition-recombination plan (DR-plan)* of the constraint *graph*. In the process of combinatorially constructing the DR-plan in a bottom up manner, at stage i, it locates a wellconstrained 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}.

Although recursive decompositions were used for geometric solving from the very beginning, DR-plans and their properties were formally defined for the first time in [**HLS01a**]. See Figures 3, 2, 4. Formally, a DR-plan of a constraint graph G is a directed acyclic graph (dag) whose nodes represent 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. There could be many DR-plans for G. See Figures 10 and 11. An *optimal* DR-plan is one that minimizes the maximum fan-in. 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).

Besides solving efficiency purposes, it is straightforward that a DR-plan is required for the classification problem (the output requirement (1) above) and for the underconstrained detection and completion problem (output requirement (4) above). In addition, it will be clear from Sections 8 and Section 7, that it is indispensable also for navigation, dealing with overconstraints and for efficient updates (output requirements (2), (3) (5) above).

A few other properties of DR-plans are of interest. We would like the *width* i.e, *number* of clusters in the DR-plan to be small, preferably 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. In addition, it is desirable for DR-plans to have the *cluster minimality* property. I.e., the child clusters of any given cluster are guaranteed to be maximal proper sub-clusters that form a minimal

covering set for the cluster – this property is used both in determining rigidity even using pure dof analysis [**LS04**], and for building stable algebraic systems of low complexity corresponding to the clusters [**SAZK04, SPZ04**].

A byproduct of the augmented definition of cluster of [**SZ04b**], described in Section 1 and discussed in Section 4 is that a somewhat modified definition of a DR-plan is used that incorporates another partial order called the *solving priority order*, which is consistent with the DR-plan's dag order, but could be more refined. The intent is that clusters that appear later in the order are not self-contained, but *dependent* and need to be solved after the clusters that appear earlier. These clusters become wellconstrained clusters only in the transformed constraint system after earlier clusters in the solving order are already solved. This type of priority order can be leveraged for assembly constraint systems, or to express procedural histories where later features are dependent upon geometric objects that are not initially present, but are the results of operations applied to earlier features.

Another important, related property is that the DR-plan *incorporate an input partial decomposition*. I.e., given an input dag P whose nodes are subgraphs of G and whose edges represent containment, a DR-plan of every node in P should be embedded in the output DR-plan for G. This is discussed in [**SZ04a**].

All properties defined above for DR-plans transfer as performance measures of the *DR-planners* or DR-planning algorithms. It is shown in [**LS04**], that the problem of finding the optimal DR-plan of a constraint graph is NP-hard, and approximability results are shown only in special cases. Nonapproximability results are not known. See Section 9. However, most DR-planners make adhoc choices during computation (say the order in which vertices are considered) and we can ask of 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.

2.2. A geometric constraint solver's phases, solving options, modes and settings. As discussed in Section 1, an effective constraint solver has a *planning* phase, which is purely combinatorial, mainly graph theoretic resulting in a DR-plan; and a *solving* phase where the polynomial systems corresponding to the nodes of the DR-plan are actually solved using an algebraic-numeric solver. In the latter phase, two solving options should be available. The *autosolve* option in which the constraint solver searches and displays a solution or realization of the entire input system (in case it is wellconstrained) There could be exponentially many such solutions, the process is not steered by the user and each solution is displayed as it is found. See for example the full solution displayed in 8. In the *navigate* option, the user should be permitted to guide the process closely by picking one of the available solutions (displayed on a separate window) for each subsystem appearing as a node in the DR-plan, which should include all the maximal well or well-overconstrained subsystems of the user's input conceptual feature, part or subassembly hierarchy. Backtracking starting at any subsystem of the DR-plan should be allowed, and partially solved systems should be displayed. See Figures 10, 20, 21, 22, 23, 24. A constraint solver's *generate* mode is used when the input is new; the *update* mode is used when incremental changes are being made to any of the part of an earlier input; the underlying algorithms of the planning and solving phases permit efficient updates for the changed input. See Figure 28. One important advantage of modularizing the phases is that in the update mode, the DR-planning phase is entirely

omitted when, for example, only value changes are made to existing constraints – these do not generically change the DR-plan. A constraint solver's *offline* setting would be used if the entire input is constructed (sketched using the GUI) before the planning and solving phases are begun. The *online* setting is used when an instant resolution of each constraint is required even as it is being input, typically by sketching on the GUI screen. See Figure 29.

Note. This manuscript is illustrated by screenshots of the FRONTIER constraint solver's sketcher. FRONTIER is an opensource 3D geometric constraint solving software suite developed by the author's group at the University of Florida.

3. The Decomposition-Recombination (DR) planning problem

Although decomposition algorithms based on constraint graphs have been proposed since the early 90's, [**FH97, Owe91, Owe93, Owe**] [**LM96, MR97**], [**Pab93, AJM93**]. prior to [**HLS01a**], the DR-planning problem and appropriate performance measures for the planners were not formally defined.

In this section, we give a table comparing 3 main types of DR-planners, with respect to the input-output requirements of Section 2 and expand on the latest of the 3 DR-planners, [**HLS01b, HLS99, LS04, OSMA01, SAZK04, SZ04b, Sit04b, Sit04a, SZ04a, SPZ04**] the Frontier vertex Algorithm (FA) which which was designed specifically to excel simultaneously in these strongly competing requirements.

3.1. Two early DR-planners.

We concentrate on two primary types of early algorithms for constructing DR-plans using constraint graphs and geometric degrees of freedom.

Note. We leave out constraint solving methods such as [**CH88, Hav91, Hsu96**] not only because they do not fit the graph based DR-planner description but also since they are nondeterministic and rely on symbolic computation, or they are randomized or generally exponential and based on exhaustive search.

The first type of algorithms, which we call SR for *constraint Shape Recognition* (e.g.[**FH96b, Owe91, Owe93, BFH$^+$95**], [**HV94, HV95, FH96b, FH97**]), concentrates on recognizing specific wellconstrained subgraphs of known shape, most commonly, patterns such as triangles. Such graphs are called *triangle decomposable*. The second type, which we call MM for *generalized Maximum Matching* (e.g. [**AJM93, Pab93, LM96, Kra92**], [**TW85, Hen92**]) is based on first isolating certain wellconstrained subgraphs by transforming the constraint graph into a bipartite graph and finding a maximum generalized matching, followed by a connectivity analysis to obtain the DR-plan. This is related to rigidity matroid based methods (generally only for 2D constraint systems with points and distances) as discussed briefly in 1, see [**GSS93**]. measures defined in the previous section.

Informally, one inherent drawback of the SR algorithms is their inability to perform a generalized degree of freedom analysis. For example, SR would require an infinite repertoire of patterns (common ones are listed in [**HV94, HV95**], [**FH96a**]). Similarly, a decomposition of underconstrained constraint graphs into wellconstrained components is possible for SR algorithms but only subject to the pattern limitations. In many cases, MM algorithms do not satisfy cluster minimality, i.e., will output DR-plans with larger, nonminimal subgraphs S_i that may

contain smaller wellconstrained subgraphs. This affects the best-choice approximation factors adversely. Most significantly, neither is able to incorporate input partial decompositions, deal with algebraically overconstrained subgraphs that often arise in 3D (or in 2D in the presence of angle constraints), or give a *complete* decomposition of underconstrained graphs into maximal clusters.

Below is a formal performance table of SR, MM and the FA (Frontier vertex) algorithms – choosing a representative algorithm (typically the best performer) in each class – using the performance measures given above. The FA DR-planner is sketched later.

3.2. Comparing performance.

Performance measure	SR	MM	FA
Generality	No	Yes†	Yes
Underconstrained decomposition	No(Yes*)	No	Yes
Incorporate Design decomposition	No(Yes$^{*,\circ}$)	No	Yes
Cluster minimality	No(No*)	No	Yes
Best-choice approximation factor	0 ($O(\frac{1}{n})^{*}$)	$O(\frac{1}{\sqrt{n}})$	$O(\frac{1}{2})$
Complexity(all have $O(n)$ width)	$O(s^2)^{*}$	$O(n^{D+1}s)$	$O(n^3 s)$

Note. The variable s in the complexity expressions denotes the number of vertices, n, plus the number of edges, m, of the constraint graph. Recall that the constant D refers to the number of degrees of freedom of a rigid object in the input geometry. ♢

The superscript '*' refers to narrow classes of DR-plans: those that require the clusters to be based on triangles or a fixed repertoire of patterns. The superscript '†' refers to general 2D systems, but not 3D. The superscript '∘' refers to a strong restriction on the design decompositions that can be incorporated into DR-plans by SR.

3.3. The Frontier Vertex DR-plan (FA DR-plan). Intuitively, an FA DR-plan is built by following two steps repeatedly:

1. *Isolate* a cluster C in the current graph G_i (which is also called the *cluster graph* or *flow graph* for reasons that will be clear below). Check and ensure a complete, maximal, dof rigid decomposition of C.
2. *Simplify* C into $T(C)$, transforming G_i into the next cluster graph $G_{i+1} = T(G_i)$ (the recombination step).

3.3.1. *Isolating Clusters.* The isolation algorithm, first given in [**HLS97a, HLS98a**] 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 12, 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

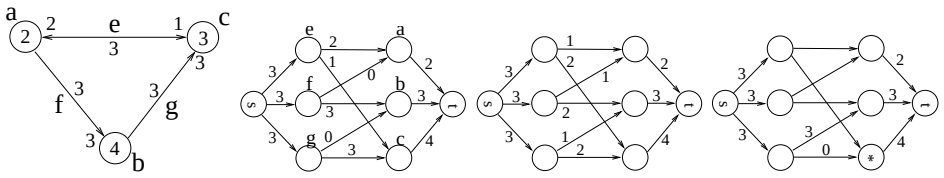

FIGURE 12. From Left. Constraint graph G with edge weight distribution. D is assumed to be 0 (system fixed in coordinate system); A corresponding flow in bipartite G^*. Another possible flow. Initial flow assignment that requires redistribution

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, [**HLS97a, HLS98a**] give an efficient algorithm to find a minimal (non-trivial, if one exists) dof cluster inside it. Then [**LS04**] gives a method to ensure a complete, maximal, dof rigid decomposition of C.

The found cluster then needs to be checked for possible dof misclassification due to the presence of constraint dependences and rotational symmetry as described in Section 1 and [**SZ04b**]. An intuitive and simplified version of these checks are described in Section4.

3.3.2. *Cluster Simplification.* This simplification was given in [**HLS01b, HLS99**]. The found cluster C interacts with the rest of the constraint graph through its *frontier vertices*; i.e., the vertices of the cluster that are adjacent to vertices not in the cluster. The vertices of C that are not frontier, called the *internal vertices*, are contracted into a single *core* vertex. This core is connected to each frontier vertex v of the simplified cluster $T(C)$ by an edge whose weight is the the sum of the weights of the original edges connecting internal vertices to v. Here, the weights of the frontier vertices and of the edges connecting them remain unchanged. The weight of the core vertex is chosen so that the density of the simplified cluster is $-D$, where D is the geometry-dependent constant. This is important for proving many properties of the FA DR-plan: even if C is overconstrained, $T(C)$'s overall weight is that of a wellconstrained graph, (unless C is rotationally symmetric and trivial, in which case, it retains its dof or weight). Technically, $T(C)$ may not be wellconstrained in the precise sense: it may contain an overconstrained subgraph consisting only of frontier vertices and edges, but its overall dof count is that of a wellconstrained graph.

Figure 13 illustrates this iterative simplification process ending in the final DR-plan of Figure 11.

The graph transformation performed by FA is described formally in [**HLS01b, HLS99**] using *simplifier maps* that provide the vocabulary for proving certain properties of FA that follow directly from this simplification mechanism. However, other properties of FA require details of the actual DR-planner that ensures them. The FA algorithm was developed in stages; it and the data structures for its implementation can be found in [**Sit04b, OSMA01, LS04, Sit04a, SZ04a, SZ04b**]. The challenge met by FA is that it provably meets the several competing requirements

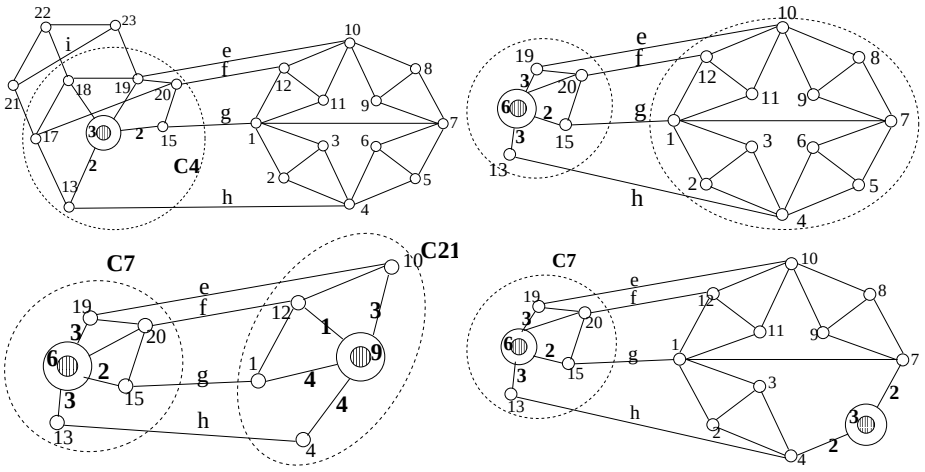

FIGURE 13. From left: FA's simplification of graph givin DR-plan in Figure 11; clusters are simplified in their numbered order: C4 is simplified before C7 etc.

of the table above. Some of these requirements are discussed in the next section. Others, such as incorporating an input feature hierarchy [**SZ04a**], are omitted.

3.4. The Frontier Vertex DR-planner (FA DR-planner). Note: a detailed pseudocode of the FA DR-planner (the existing version, as well as incorporating the module-rigidty algorithm of this paper) can be found in [**Sit04b**], [**Sit04a**]. The pseudocode has been implemented as part of the downloadable, opensource FRONTIER geometric constraint solver [**OSMA01**], [**Sit04b**], [**Sit04a**], [**Sit04c**].

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 cluster minimality - this is needed to ensure true clusters even just according to the dof-based definition of Section 1;
(b) for underconstrained graphs: outputing a complete set of maximal clusters as sources of the DR-plan;
(c) controlling width of the DR-plan to ensure a polynomial time algorithm.

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

The method *DistributeVertex* distributes all edges (calls DistributeEdge) connecting the current vertex to all the vertices considered so far. When one of the edges cannot be distributed and a minimal dense cluster C is discovered, its simplification $T(C)$ (described above) transforms the flow graph. The flows on the internal edges and the core vertex are inherited from the old flows on the internal edges and internal vertices. Notice that undistributed weights on the internal edges simply disappear. The undistributed weights on the frontier edges are distributed (within the cluster) as well as possible. However, undistributed weights on the frontier edges (edges between frontier vertices) may still remain if the frontier portion of the cluster is severely overconstrained. These have to be dealt with carefully. (See discussion on dealing with the problems caused by overconstraints in Section 7.) The new cluster is introduced into the DR-plan and the cluster queue.

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

Eventually, when the cluster queue is empty, i.e, all found clusters have been distributed, the DR-plan's top level clusters are guaranteed to be the complete set of maximal dof rigid subgraphs of the input constraint graph. See [**LS04**] for formal proofs.

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

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

3.4.1. *Ensuring Cluster Minimality.* First we intuitively explain why cluster minimality is a crucial property. In Figure 14, after C_1 and C_2 are found, when C_1 is distributed, C_1 and C_2 would be picked up as a cluster, although they do not form a cluster. The problem is that the overconstrained subgraph W intersects C_1 on a trivial cluster, and W itself has not been found. Had W been found before C_1 was distributed, W would have been simplified into a wellconstrained subgraph and this misclassification would not have occurred.

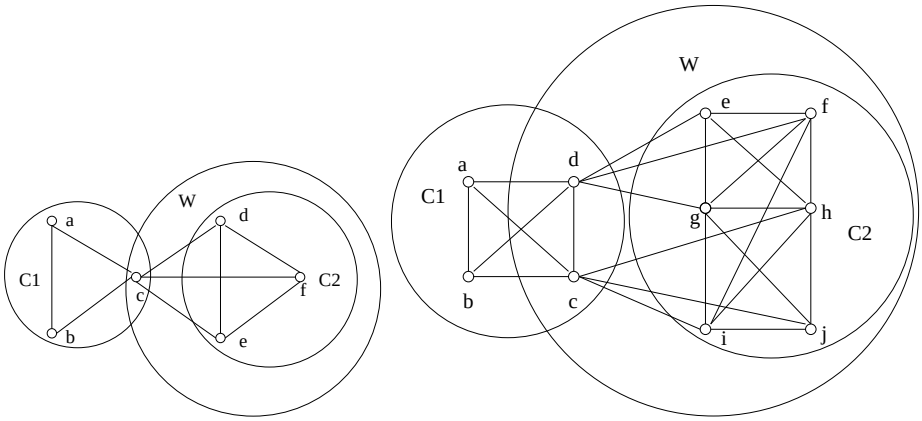

FIGURE 14. Finding W first will prevent dof misclassification: Left 2D example, Right 3D example.

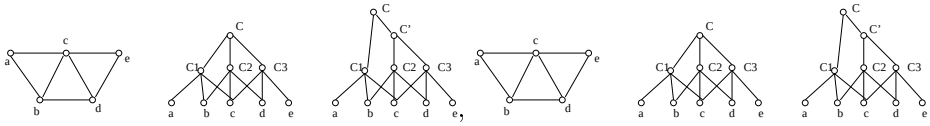

FIGURE 15. Ensuring Cluster Minimality: E is a set of essential clusters that must be present in any subset of the children of C that form a cluster. In this case, E itself forms a cluster. C' is a cluster made up of a proper subset of at least 2 of C's children

It has been shown in [**LS04**] that this type of misclassification can be avoided (W can be forced to be found after C_2 is found), by maintaining three invariants. The first two are described here. The third is highly related to property (b) and is described in the next subsection.

The first is the following invariant: always distribute all undistributed edges connecting a new found cluster C (or the last distributed vertex that caused C to be found), to all the vertices distributed so far that are outside the cluster C. Undistributed weight on edges inside C are less crucial: if they become internal edges of the cluster, then this undistributed weight "disappears" when C is simplified into a wellconstrained cluster; there is also a simple method of treating undistributed weight on frontier edges so that they also do not cause problems - the method and proof can be found in [**LS04**]).

The second invariant that is useful for ensuring cluster minimality is that for any cluster in the DR-plan, no proper subset of atleast 2 of its child clusters forms a cluster. FA ensures this using a generalization of the method *Minimal* of [**HLS97a, HLS98a**] which finds a minimal dense subgraph inside a dense subgraph located by *DistributeVertex* and *DistributeEdge*.

See Figure 15. 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 atleast 2 C_j's forms a cluster C', then the clusMin algorithm is repeated inside C' and thereafter in C again,

replacing the set of child clusters of C that are inside C' by a single child cluster C'. If instead no such cluster is found, then the removed cluster C_i the *essential*. I.e., it belongs to every subset of C's children that forms a cluster. When the set of clusters itself forms a cluster E (using a dof count), then *clusMin* is called on C again with a new child cluster E replacing all of C's children inside E.

3.4.2. *(b) Finding a complete set of maximal clusters in underconstrained graphs.* While the DR-planner described so far guarantees that at termination, top level clusters of the DR-plan are maximal. It also guarantees that the original graph is dof underconstrained only if there is more than one top level cluster in the DR-plan. However, in order to guarantee that *all* the maximal clusters of an underconstrained graph appear as top level clusters of the DR-plan, we use the observation that any pair of such clusters intersect on a subgraph that reduces (once incidence constraints are resolved) into a trivial subgraph (a single point in 2D or a single edge in 3D). This bounds the total number of such clusters and gives a simple method for finding all of them. Once the DR-planner terminates with a set of maximal clusters, other maximal clusters are found by simply performing a *Pushoutside* of 2 units on every vertex (in 2D) or every vertex and edge (in 3D), and continuing with the original DR-planning process until it terminates with a larger set of maximal clusters. This is performed for each vertex in 2D and each edge in 3D which guarantees that all maximal clusters will be found. See [**LS04**] for proofs.

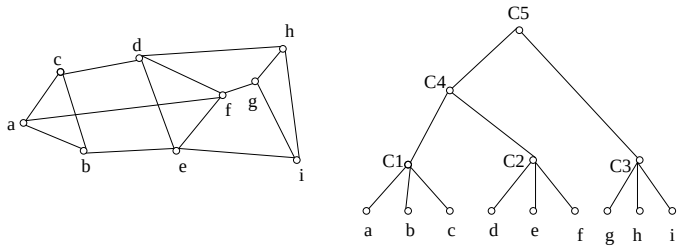

FIGURE 16. Prevent accumulation of clusters

3.4.3. *(c) Controlling width of the DR-plan.* FA achieves a linear bound on DR-plan width by maintaining the following invariant of the cluster or flow graph: *every pair of clusters in the flow graph (top level of the DR-plan) at any stage intersect on at most a trivial subgraph.* FA does this by repeatedly performing 2 operations each time a new potential cluster is isolated.

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

For the example in Figure 16, when the DR-plan finds the cluster C_2 after C_1, the DR-planner will find that C_1 can be enlarged by C_2 The DR-planner forms a new cluster C_4 based on C_1 and C_2 and puts C_4 into the cluster queue, instead of putting C_2 to cluster queue.

The second operation is to iteratively *combine* $N \cup D_1$ with any clusters $D_2, D_3, \ldots$ based on a nonempty overlap that is not rotationally symmetric or trivial. In this

case, $N \cup D_1 \cup D_2$, $N \cup D_1 \cup D_2 \cup D_3$ etc. enter the DR-plan as a staircase, or chain, but only the single cluster $N \cup D_1 \cup D_2 \cup D_3 \cup \ldots$ .. enters the cluster graph after removing D_1, D_2, D_3 ....

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

4. Correcting the 3D dof misclassifications

As pointed out in Section 1 (see also Section 9), it is not known despite several conjectures and heuristics in the combinatorial rigidity literature [**GSS93**], whether a tractable, purely combinatorial analysis exists to correctly classify a 3D constraint system (or 2D constraint graph with other constraints besides distance constraints) as generically rigid. The few constraint solvers that deal with 3D systems for example the proprietary, commercial D-cubed solver [**Owe**], which handles certain 3D constraint systems such as the "pipe routing problem" and FRONTIER [**OSMA01, Sit04b, Sit04a, LS04, SZ04b**], which deals with fairly general 3D systems, must grapple with this problem in some systematic fashion.

In this section, we discuss an intuitive simplification of a method used by the FRONTIER constraint solver [**SZ04b**] that uses a concept called module-rigidity to rectify dof misclassification of a large class of 3D constraint graphs. Rotationally symmetric, trivial subgraphs and clusters underlie these misclassifications. Below we discuss both their *direct* and *indirect* involvement in such misclassifications and how to rectify them - the latter is especially nontrivial (see Figures in Section 1 concerning the "bananas" problem) and both are complicated by the presence of incidence constraints.

Methods for handling other common misclassifications discussed in 1 caused by 2D angle constraints are discussed only briefly in Section 8, under the context of online solving.

4.1. Rotationally symmetric, trivial subgraphs. Some common 3D examples of rotationally symmetric, trivial subgraphs are: a single point in 3D or 2D (does not cause any misclassifications); 2 points in 3D incident on a line; 2 points in 3D and a distance constraint between them, along with other incident points, line segments and lines, reducing to a trivial fixed-length line segment cluster. Here are some examples of dof analysis problems *directly* involving the latter two types of subgraphs and how they are rectified by the FRONTIER constraint solver's FA-based DR-planner [**Sit04b, Sit04a, SZ04b**].

4.1.1. *Detecting rotational symmetric and underconstrained trivial subgraphs.* For the 3D example in Figure 17, the points a and b are incident to the line l. The whole object has 6 degrees of freedom. It would be deemed a cluster in the usual dof analysis. However, it is underconstrained and should not be picked up as a cluster. Intuitively, the object loses one degree of freedom because of its rotational symmetry - a *geometric* property, which compensates for its underconstrainedness, thereby "fooling" the simple dof count into classifying it as wellconstrained.

To solve this problem, in 3D, after a new subgraph is found to be a cluster by dof analysis by the DR-planner, it checks whether it is a trivial, rotationally symmetric subgraph. Detecting this involves organizing potentially numerous incidence constraints: these are divided into 3 groups, those objects that are incident on the left endpoint, the right endpoint and on both. Once the rotational symmetry is detected, if this subgraph has 6 degrees of freedom, then it is not a cluster and should

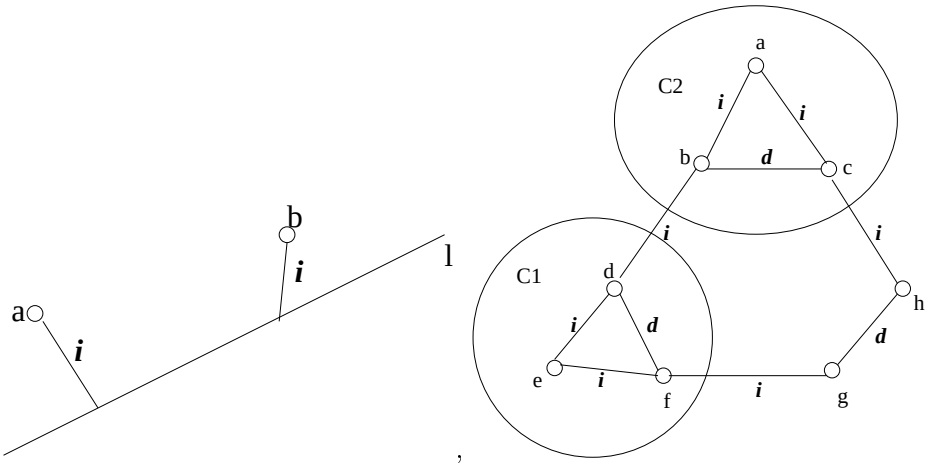

FIGURE 17. Rotationally symmetric object: detecting by organizing incidences (left); special case of FA simplification (right); d denotes distance and i incidence constraint

be discarded. If it has 5 or fewer degrees of freedom (reducing to a fixed-length line segment in 3D), it is kept as a cluster and marked as rotationally symmetric, as a crucial help during navigation of its solution space by as discussed in Section 5.

4.1.2. *Enlarging and combining clusters based on nontrivial overlap.* One of the important features of any general, tractable DR-planner that avoids a combinatorial explosion is that it restrict the width or number of clusters in the DR-plan. For this purpose, we now show that the check for rotational symmetry - by organizing incidences - given in subsection 4.1.1 is crucial.

As described in Section 3, 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 cluster graph at any stage intersect or overlap on at most a rotationally symmetric or trivial subgraph. This is done by enlarging and combining clusters based on nontrivial overlap.

For the example in Figure 18, the overlapped part of $D_1 = C_1$ and $N = C_2$ has a DR-plan which has 2 source clusters, *ab* and *cd*. Once the incidences are organized, the overlap is seen to be a trivial rotationally symmetric subgraph (albeit not a cluster) and hence N would *not* be enlarged by D_1.

In one special case, clusters should be enlarged even when their overlap is a trivial rotationally symmetric cluster. For example, as in Figure 18, either N or D_1 is a rotationally symmetric cluster and they overlap on a rotationally symmetric cluster, then N can in fact be enlarged by D_1 (even without any other constraint between them). Here, $N = C_2$ is a rotationally symmetric cluster and can be enlarged by $D_1 = C_1$.

4.1.3. *Rotational symmetry and overconstraints.* For the 3D example in Figure 17 (right) the line segment C_1, with 5 degrees of freedom, is an overconstrained cluster based on dof. To retain many of their properties, FA and other DR-planners would have to simplify such clusters (see Section 3) into a wellconstrained cluster with 6 dof's. In this case, however, such a simplification would cause FA or any

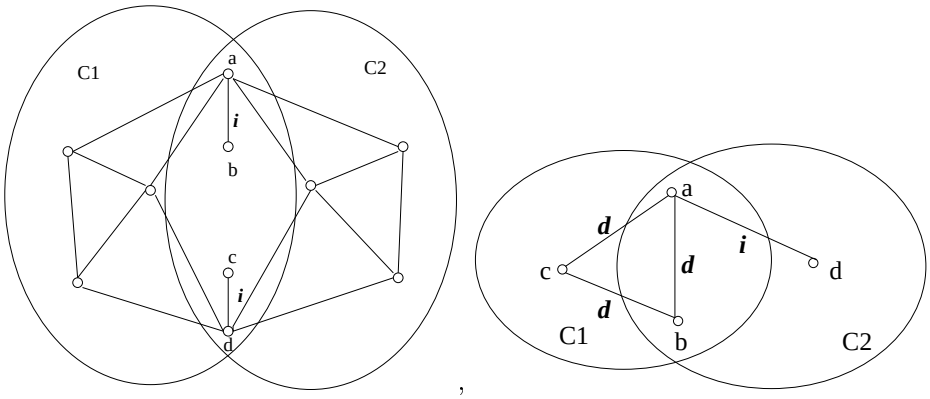

FIGURE 18. Enlarging clusters: organizing incidences to (left) detect rotationally symmetric subgraphs in overlap and (right) detect a rotationally symmetric participating cluster

standard dof analysis to fail to find this simple triangle as a cluster! Since linesegments and triangles are basic building blocks of constraint systems, this problem needs to be corrected.

To solve this problem. when the DR-planner detects a rotationally symmetric cluster with 5 degrees of freedom, it not only marks it as rotationally symmetric, but simplifies it into cluster that maintains its 5 degrees of freedom. Intuitively, we view the loss of 1 dof due to rotational symmetry as yet another type of overconstraint a *geometric* overconstraint (distinct from the combinatorial and constraint dependences discussed in Section 5). These clusters behave in many ways like wellconstrained even though they possess 1 less degree of freedom.

4.2. The "Bananas" Problem. We now discuss a classic case where dof analysis fails to correctly classify a large class of constraint graphs due to the constraint dependence caused by the *indirect* involvement of rotationally symmetric, trivial clusters or subgraphs. Some example figures illustrating this were given in Section 1. These errors must be checked for as soon as a potential cluster is isolated by flow during the FA DR-planning process. We briefly describe an intuitive simplification of how the FRONTIER constraint solver ([**Sit04b, SZ04b**]) handles these errors, thereby correctly classifying a large class of graphs for which a pure dof analysis is inadequate. As mentioned in Section 1, complete process involves ensuring an augmented notion of rigidity called module-rigidity and can be found in [**SZ04b**].

4.2.1. *Problem Description.* For the 3D example in Figure 5, if the DR-planner operates purely by dof analysis, it will locate this graph as a cluster. In fact, the graph is generically overconstrained since the distance between the 2 overlap points in the center is determined independently both by the left and right clusters. The graph (right) however is combinatorially underconstrained: *If* these distances were consistent (equal), then the graph would be underconstrained. It has a dof of 6 (appears to be wellconstrained) because its underconstrainedness is compensated by its overconstrainedness.

The example in Figures 7, 6 is very similar to the above example, except that it has a distance constraint between the two overlap points. In this case, the

left and right clusters are combinatorially overconstrained and will be simplified into well-constrained clusters. The combined cluster will not be picked up by dof analysis unless there is one more constraint between the two clusters, say a distance constraint as in Figures 7, 6 (left). In both of these cases, the dof classification is in fact correct.

In fact, as pointed out in Section 1, the constraint system has constraint dependences, if the common overlap O of *any* subset of its wellconstrained clusters is underconstrained; the undetermined edges in the overlap are independently determined by each of the clusters, which is the source of the overconstraint - i.e, many equations with dependent or even identical left hand sides, but generically independent right hand sides. The above "bananas" is a special case of this, where the overlap is a rotationally symmetric trivial subgraph that is not a cluster. This type of constraint dependence causes the dof analysis to be inaccurate only in this specific "bananas" case. See Claim 1 below.

Different responses are possible when an constraint dependence is detected.
(a) The user could be asked to correct the offending underconstrained overlap O by adding explicit constraints to the overlap. This would now cause explicit combinatorial overconstraints, which can be detected and corrected (see Section 7, [**HSY04**]). This could also cause new subgraphs to become self-contained clusters, where previously they would have been directly made part of larger clusters.
(b) We assume that the source of constraint dependence is intentional and known to the user who has otherwise ensured that the overconstraint is consistent (the right hand sides have the same dependence as the left hand sides of the corresponding equations). In this case, the user, after being informed of an constraint dependence, may choose not to correct it. I.e, it is desirable that all constraint dependences (atleast those within clusters located by the flow algorithm or other dof analysis) be detected, since they could cause it is sufficient to *treat* only those cases of constraint dependence that actually cause a misclassification by a dof analysis.

4.2.2. *A recursive solution.* When a new cluster C is found in 3D, we describe how it is checked for dof misclassification and how it is treated –in the case when it consists of exactly 2 child clusters which have been correctly classified. Addressing this case is sufficient because of the following 2 claims.

Claim 1: Let C be a cluster in 3D detected by flow or dof analysis. Assume C consists of child clusters that have all been correctly classified as clusters (i.e., they have passed the dof misclassification check being described here). Furthermore, assume that C satisfies the cluster minimality property described in Section 3. I.e, for every cluster in the DR-plan, no proper subset of its child clusters form a cluster. Finally, assume that C has been misclassified due to an constraint dependence caused by more than one cluster independently fixing a single distance between a pair of points not connected by an explicit distance constraint. Then C must have consisted exactly of a *pair* of child clusters forming the "bananas" structure, i.e., whose overlap is a trivial (rotationally symmetric) cluster having 6 dofs.

To prove the claim, first consider the case where C consists of a pair of child clusters – in this case, while their overlap could be an underconstrained graph with arbitrarily many clusters, it is not hard to see that a dof misclassification takes place exactly when the pair forms the "bananas structure."

Next, consider the case where cluster-minimal C consists of 3 or more clusters. Also assume some subset of its child clusters have a common overlap that is an

underconstrained system causing the source of an constraint dependence of the type described above. It is not hard to see that a dof misclassification would result exactly when some pair of clusters form the "bananas" structure. But that would imply that pair would have been picked up as cluster (according to dof analysis) contradicting the cluster-minimality of C.

In fact, the proof of the claim is further simplified in the case of the FA DR-planner: due to the enlarging and combining operations it uses to control width of the DR-plan, (see Section 3) - if the parent cluster C has more than 2 child cluster s, then for every pair of them, their overlap is guaranteed to be equivalent (via incidence constraints) to a trivial subgraph.

Claim 2: The FA DR-plan has the *cluster minimality* property. I.e, for every cluster in the DR-plan, no proper subset of its child clusters form a cluster.

Claim 2 is proven true (by [**LS04**] and Section 3) only for the FA DR-plan built by a purely flow based dof analysis. In particular, that FA DR-planner first finds (by dof flow analysis) a potentially nonminimal cluster S consisting of clusters $D_1, \ldots, D_k$ and then runs a minimality-ensuring algorithm (that recursively performs a flow based dof analysis again) and constructs a sub-DR-plan of S, i.e., rooted at S, whose leaves are $D_1, \ldots, D_k$ (see [**LS04**], Section 3). However, the FA DR-planner based on purely dof flow analysis may misclassify the minimal "clusters," thus found and thereby the entire sub-DR-plan may contain misclassified clusters, including the original nonminimal "cluster" S.

Thus, given Claim 1, the most efficient way to modify the FA DR-planner to avoid misclassifications due to this type of constraint dependence, is to perform the base case of the "bananas" correction (described below) *during* the minimality-ensuring portion of the FA algorithm. I.e., the base case is performed during the construction of the sub-DR-plan for S, assuming that the clusters $D_1, \ldots D_k$ have already been checked and corrected. I.e, it is performed each time a cluster C consisting of a *pair* of (already checked and corrected) child clusters C_1 and C_2 is located during and is being considered as an element in the sub-DR-plan for the cluster S. Once the "bananas" correction described below is completed: in the first 2 cases below, C is inserted into the sub-DR-plan for S and in the last case, C is *not inserted*. And the minimality-ensuring method continues.

At the end of the process, a sub-DR-plan for it is found, with cluster-minimal clusters corrected for dof misclassification, and either S is found to be a true cluster, in which case the sub-DR-plan has a single root or sink, namely S itself, or S is not a cluster, in which case the sub-DR-plan's roots or sinks correspond to the complete set of maximal clusters within S.

4.2.3. *The base case of bananas correction.* When a cluster is found consisting of 2 child clusters C_1 and C_2, during the the minimality-ensuring method, first check whether the entire overlap between C_1 and C_2 reduces to a rotationally symmetric subgraph with 5 or 6 dofs. (after incidences are organized as described in the previous subsections). If it is not a rotationally symmetric subgraph, we put $C = C_3$ in the DR-plan (after checking for potential enlargement described in previous sections, and checking for and announcing any constraint dependences, i.e, if the overlap is not wellconstrained).

If the overlap is a rotationally symmetric *cluster*, i.e., it reduces either to a single point (after resolving incidences) or there is a distance constraint in the

overlap, and there are explicit constraints as in Figure 6, 7, then C_3 is retained as a normal wellconstrained cluster.

If the overlap is a rotationally symmetric subgraph which is not a cluster, *and* there are other constraints between C_1 and C_2, retain C_3, but mark it as a cluster containing a constraint dependence (and announce it); it is then simplified into a cluster with 6 dofs, retained but marked to be "handled with care" during solving and navigation (Section 5).

If there are no other constraints between C_1 and C_2, then the constraint dependence must be announced. there is a dof misclassification and in fact, the found cluster $C = C_3$ should be thrown away. However we need to prevent it from being "found" again, and moreover, we need to prevent it from causing a dof misclassification of a larger cluster that includes both C_1 and C_2, and is found at a later stage.

To do so, we remove the constraint dependence by adding an edge - representing a "virtual" distance constraint - to the overlap. The following corrections need to be done:
a) all the existing clusters that contain this edge must be resimplified so that they continue to have a dof of 6. b) the edge must be appropriately flagged as "belonging" to C_1 or C_2 (its distance value was fixed by C_1 or C_2), so that a cluster D containing this edge that will be found in the future by the DR-planner (and passes the "bananas" check), have to be handled as follows: (i) if D remains a cluster on removal of this edge, then the source of a potential constraint dependence has been detected – this need not be announced until the point when it causes an underconstrained overlap between D and a partner cluster which is picked up as a future cluster. (ii) if D requires the edge to be picked up as a cluster by dof analysis, then D should inherit the flag of the edge so that during the solving phase D is solved only *after* atleast one of the original clusters C_1 or C_2 is solved (and thereby the distance associated with the edge has been fixed).
c) both the clusters C_1 and C_2 in the pair are distributed again (see description of flow method in Section 3)

Note. It is important to notice that the above method utilizes the second partial ordering or the *solving priority order* of the clusters of the DR-plan given in Section 1. Recall that this ordering is consistent with the DR-plan's inherent partial order. More significantly, it uses the modified definition of DR-DAG given in Section 1, where not every node of the DR-plan represents an independent cluster in the original graph or constraint system. A node represents a generically wellconstrained system in the *transformed system or transformed graph after* all the clusters before it in the solving order have been solved or simplified.

Figure 19 shows examples of the bananas situation that are correctly classified by the method described above.

In Figure 19 Top Left: the graph is a cluster according to dof analysis, but not really rigid, as detected by the bananas correction and as seen by the decomposition shown. Top Right: rigid, but no pair of self-contained clusters shown forms a truly rigid subgraph, as detected by the bananas correction although they do form clusters according to dof analysis. Bottom: not rigid as detected by the bananas correction, but a cluster according to dof analysis; complete maximal module decomposition and solving priority orders found by the bananas correction: the pair C_1, C_2 forms a rigid C_5 but that C_5 can be solved only after C_3 is solved; I.e.,

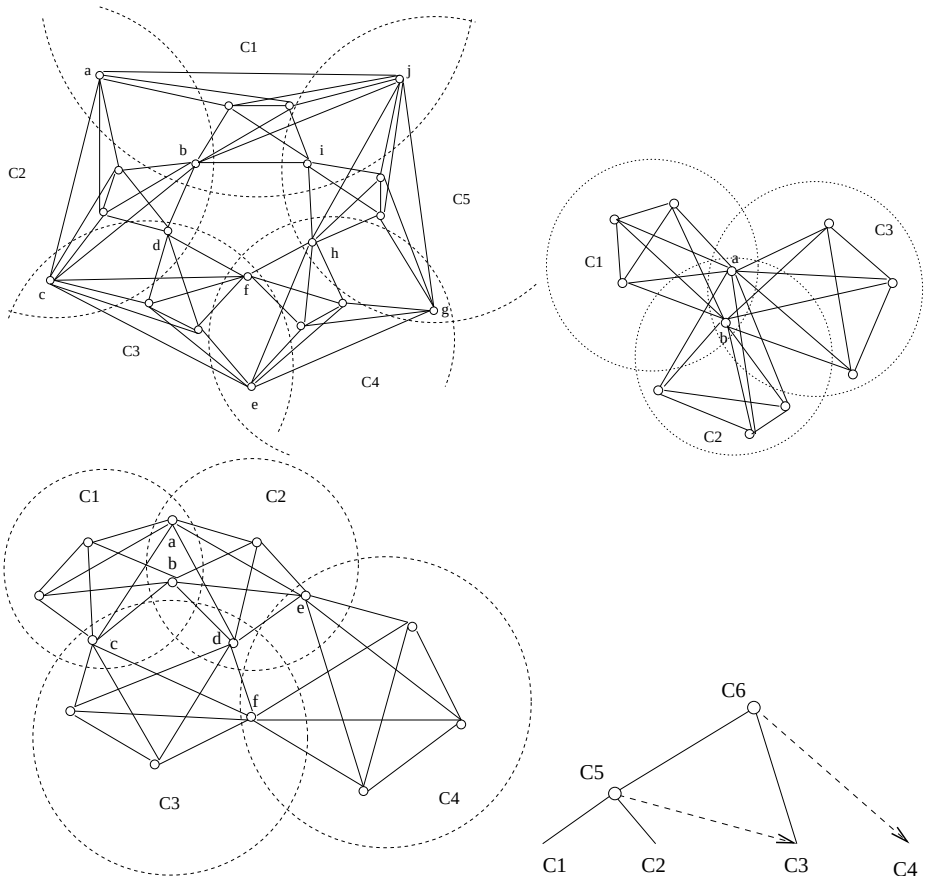

FIGURE 19. Examples where bananas correction works. See text for explanation.

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 rigid C_6, but only according to solving priority order shown. Bottom Right: DR-plan for left constraint system with 2 sources or roots: C_6 and C_4.

5. Solving and Navigating the solution space

In this section we discuss combinatorial approaches to solving and navigating the solution space of a cluster, during the solving phase of a constraint solver's operation, once the planning phase has already output a DR-plan. For underconstrained systems the situation is more complex, see Section 7, and Section 9.

A wellconstrained system may have exponentially many solutions in the number of geometric primitives. For example, even a simple ruler and compass construction in 2D permits 2 solutions for each constructed point (reflections are considered distinct since they cannot be obtained by a physical euclidean transformation such as rotation or reflection) Existing approaches to controlling this combinatorial explosion fall under three broad categories.

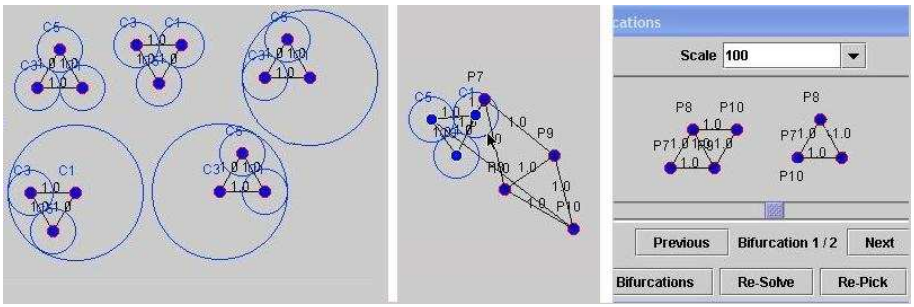

FIGURE 20. Left: display of all solutions of the cluster formed by the 3 circles in constraint system of Figure 2. Middle: partially solved sketch after choosing one of them; Right 2 solutions of the diamond of points and distances

One standard approach [**Fud95, FH96b, Owe**] attempts to automatically pick a solution that is as close as possible to the appearance of the input sketch. Typically, chirality (or relative orientation) of the geometric primitives in the sketch is used as the guide. A related approach uses explicitly defined "navigation" constraints or "declarative solution selectors" provided by the user. The formal basis for this began with [**Hen92**], and some of the more recent methods are surveyed in [**BS03**]. The use of genetic algorithms for dealing with these extra constraints are given in [**JALS03**]. In general, these constraints could include chirality or relative orientation constraints, and other constraints related to chirality (by oriented matroid theory [**BVS$^+$93**]) such as intersection or nonintersection of convex hulls or subsets of the points. They reduce to specific polynomial inequalities obtained from determinants of matrices of indeterminates. Another type of navigational constraint are the so-called relational or engineering constraints [**FH97**].

A second approach gives the user an indexing [**BFH$^+$95**], of the solution space using a Decomposition Recombination (DR) plan of the constraint system as a guide. These DR-plans are generated by most constraint solvers to efficiently guide the algebraic-numeric solvers. Specifically, they break down the large polynomial system that arises from the entire geometric constraint system into small subsystems that can be handled by algebraic-numeric solvers. This is crucial since the latter typically have exponential time complexity.

The methods of [**BFH$^+$95**] provide tractable steering for simple, 2D constraint systems that are triangle decomposable, or have linear DR-plans, similar to ruler and compass constructible systems, that permit solving for one geometric primitive at a time.

In this case, each such addition provides two "bifurcation" choices that are typically two reflections of the newly solved-for primitive. However, such DR-plans exist only for special constraint systems and in overconstrained cases they may not incorporate a user's conceptual feature decomposition, potentially denying the user's prefered navigation path. This problem is explained in detail in [**SZ04a**].

A *third* mixed approach (used by the FRONTIER constraint solver [**OSMA01, Sit04b, Sit04a, SAZK04**]), is a hybrid method that both uses navigation constraints and offers two options: a *visual walkthrough* with backtracking or *automatic search* for a solution. See Figures 20, 21, 22, 23, 24.

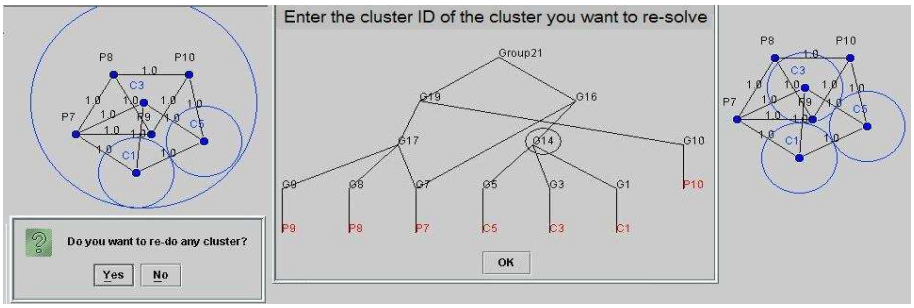

FIGURE 21. Left: complete solution of root cluster of Figure 2 after choosing one of the solutions for each of the child clusters (the 3 circles and the diamond of Figure 20). Right: new solution of root cluster after backtracking to repick a different solution of the 3 circle cluster

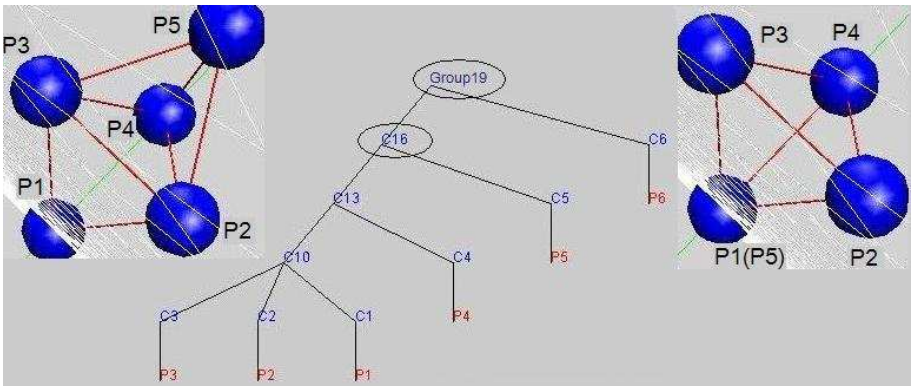

FIGURE 22. Solution navigation for constraint system in Figure 4: Two solutions for the child cluster of the root marked in the DR-plan.

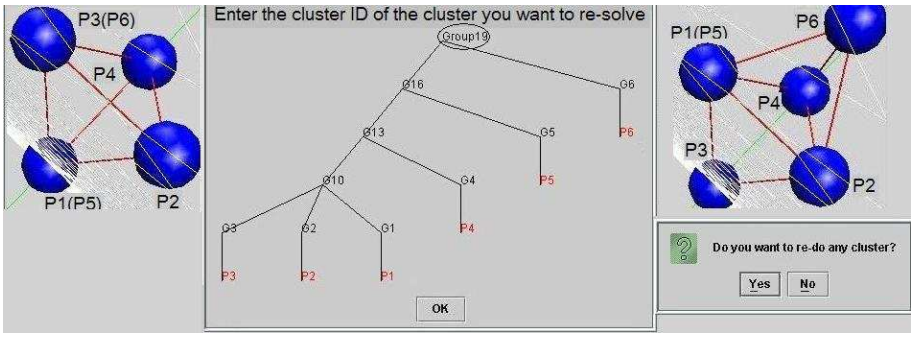

FIGURE 23. After picking the solution on right in Figure 22, 2 final solutions for root cluster with backtracking option.

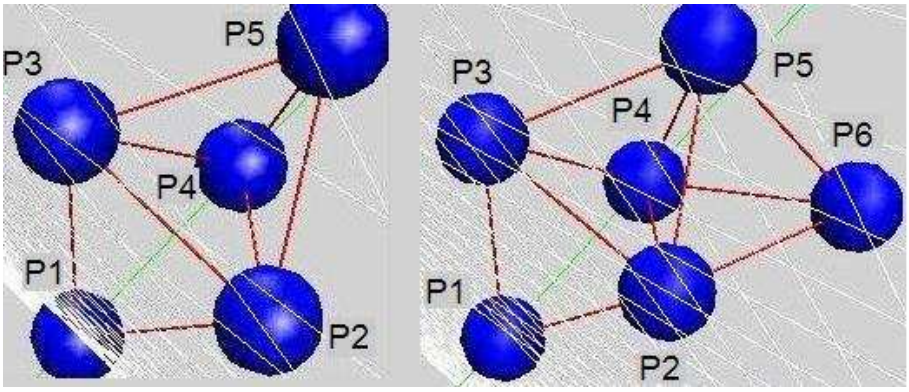

FIGURE 24. After backtracking and repicking the left solution of child cluster in Figure 22, two final solutions for root cluster (complete system of Figure 4).

Furthermore, the walkthrough is based on *general* DR-plans (for 2D or 3D systems), such as the FA DR-plan (output of the Frontier vertex Algorithm) of Section 3, that incorporates a user's desired conceptual feature or subassembly decomposition, which is crucial for a user-directed search for the solution space.

The method described completely in [**SAZK04**] has the advantage that it can be a stand-alone module: it takes as input the constraint graph, the DR-plan (output of *any* DR-planner) and navigation constraints, and outputs to the GUI (resp. stores in the DR-plan structure) the realizations or solutions - of the subsystems encountered during navigation. Other interactive input includes the user's choice of solutions to the clusters. The method provides fully flexible *backtracking* for redoing such choices starting from *any* cluster. See Figure 28. In the FRONTIER constraint solver, this method is called the *Equation and Solution Manager (ESM)*. [**OSMA01, Sit04b, Sit04a, SAZK04**]. The module in addition efficiently deals with updates or changes to the constraint graph and the navigation constraints; these are discussed in Section 8.

There are 4 crucial parts of the ESM. The first is the overall control flow based on a standard recursive traversal of the DR-plan and an efficient datastructure for maintaining chosen realizations, which maintains clarity and controls complexity during navigation, and provides fully flexible backtracking or *steering*. See [**SAZK04, OSMA01, Sit04b, Sit04a**] for descriptions of this part of the ESM. The second part is a method for building independent algebraic systems corresponding to the clusters of the DR-plan. The third part is a purely combinatorial method [**SPZ04**] for optimizing the algebraic complexity of these systems - this method interestingly uses minimum spanning trees on a graph obtained from the cluster structure. The weights are determined based on a cluster solving method that uses rigid body mechanics and quaternions. The fourth part leverages the modularity of the ESM as well as the above characteristics to incorporate navigation constraints.

Note. We restrict our description here to the second and the fourth parts described above.

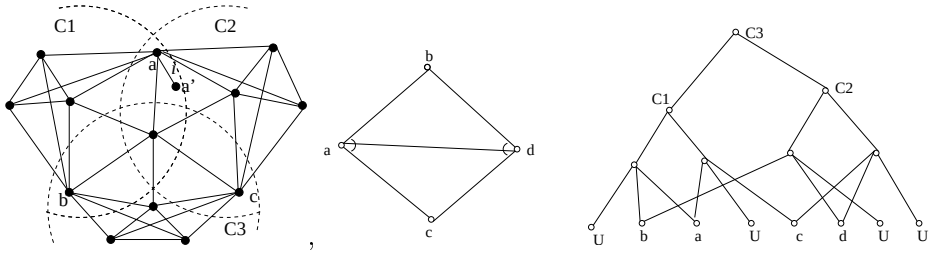

FIGURE 25. Left:Assigning overlap equations. Right:Dealing with constraint dependences during solving.

6. Building algebraic systems - the stability issue

The polynomial system corresponding to a cluster C typically has 9 variables (entries of the rotation/translation matrix in homogeneous coordinates) corresponding to *each* child cluster of C. Three of these variables are dependent, via equations of the type $s^2 + c^2 = 1$, where s and c represent the sine and cosine of one of the rotation angles. These could be reduced using a quaternion or other forms for optimization of algebraic complexity [**SPZ04**] which is a separate problem that does not influence our discussion here. In addition there are polynomial equations for the constraints that relate primitive geometric elements of the child clusters. These are the constraints associated with C and need to be resolved to obtain a solution to C given the solutions of its children. The solutions of the children determine the relative positions of the geometric elements within the child clusters: these appear on the left hand sides of the polynomial system for C. But in the cases where C appears after one of its children (or some other cluster) in the solving priority order (see Section 2) one or more of C's distance constraints (values) could be determined by the solution of the child clusters (or other clusters). These appear on the right hand sides of the polynomial system corresponding to C. (These equations could result in an undesirable compounding of numerical error, which is outside the scope of our discussion here).

In order to avoid even more basic instabilities, it is usually necessary to remove overconstraints and dependent constraints: a stable, wellconstrained system generically has as many variables as equations. The number of equations could exceed the number of variables, because of explicit overconstraints that appear as edges in the constraint graph or because of overconstraints as in the "bananas" case, where some of the constraints are dependent. The former can be removed using the method of [**HSY04**], which detects and removes so-called *reducible* edges whose removal does not make the cluster underconstrained. The latter are explicitly marked by a good DR-planner such as [**SZ04b**] as discussed in Section 2: the method of dealing with these is discussed later in this section.

The bulk of this section, however, deals with wellconstrained subsystems with no constraint dependences. Even for these, picking an independent set of algebraic equations is a nontrivial task. Some of these equations are incidence or overlap equations (equating 2 variables) caused either by explicit incidence constraints or by geometric elements that are shared by two or more overlapping child clusters.

Figure 25 illustrates the problem for an example in 3D, the entire cluster C has solved child clusters C_1, C_2, C_3. To solve C, i.e, to solve for the rotation/translation

of C_2 and C_3, say, into the coordinate system of C_1 (chosen to be the coordinate system for C) several overlap equations, that require that, for instance, point A in C_1 fall in the same location as point A in C_2. We treat these as distinct from explicit incidence constraints, which directly enter the dof analysis.

For generic stability (independence), when m child clusters are present in the parent cluster, the number of equations to solve the parent cluster be at most $D(m-1)$ (D is 3 in 2D and 6 in 3D). This includes the number of equations caused by any other constraints between C_1 and C_2 (in this example, there are none) and the overlap constraints.

Even in wellconstrained parent clusters C although the dof analysis of the cluster C (appropriate inclusion-exclusion count of the dofs of the child clusters and their overlaps) would show the the cluster to be wellconstrained, the number of potential overlap equations could exceed the $D(m-1)$ as in the above example: each of the points A, A', B, and C generates 3 overlap equations and the point D generates 3*2 = 6, totaling 18 overlap equations which exceeds $D(m-1) = 6*2 = 12$.

The ESM uses a method (see the *getEquation* and *getOverlapEquation* methods of the ESM pseudocode in [**Sit04a, Sit04b**]) that partitions overlaps and other constraints between the child clusters of C in a way that they constitute a stable and and algebraically independent set of $D(m-1)$ constraints for solving C.

We decide on which overlap equations to discard as follows. Let n be the number of other (nonoverlap) constraints relating the child clusters of the parent cluster C being solved. If C is wellconstrained, then $D(m-1) \geq n$. To choose and partition the overlap equations, we stipulate the following.

- Any cluster (including trivial clusters) in the overlapped region should participate in atmost as many overlap constraints as the number of its degrees of freedom, although the total number of position variables of its geometric primitives could be higher; For example, in 3D, any line segment (trivial cluster), even one that is of variable length should participate in at most 5 overlap equations although its two end points constitute 6 position variables, since they belong to a cluster and hence the distance between them has been fixed by that cluster (this is the constraint dependence or overconstraint caused by the classic bananas structure of Section 1). any underconstrained overlap is a source of constraint dependences and overconstraints which must be eliminated in building a stable system, regardless whether it causes a misclassification during dof analysis.
- Any coordinate position of a geometric element in a cluster (an expression in the rotation/translation variables of the cluster) should participate either in an explicit incidence constraint relating child clusters of C or an overlap constraint, but not both; otherwise when we set everything else except that variable to constants, we get either 2 identical equations, or 2 inconsistent equations with the variable occuring with the same coefficient, i.e, 1;
- If two geometric elements participate in an explicit incidence constraint within a cluster C_i, then only one of them can participate in an overlap equation;
- Any non-rotationally symmetric child cluster should participate in at least D (3 in 2D; 6 in 3D) equations (overlap and nonoverlap combined); for points

in 2D, this bound is 2, and for fixed length line segments in 3D, this number is 5.
- Any pair of clusters should be related by at most D overlap or nonoverlap equations (this number is less if $m >= 2$, or if one of the clusters is rotationally symmetric).
- Total of $D(m-1) - n$ overlap equations should be chosen.

It can be shown that for generically wellconstrained cluster C, *any* assignment or partitioning algorithm for choosing the overlap equations results in a generically stable and independent system, provided it satisfies these criteria.

As mentioned in Section 3 a significant advantage of using an FA DR-planner in this context is the following. In case C consists of a just a pair of child clusters, then their overlap could be an underconstrained graph with arbitrarily many clusters, but ensuring the first requirement above is relatively easy, by simply obtaining the DR-plan of the overlapped part. If, on the other hand, the parent cluster C has more than 2 child clusters, then because of the cluster-minimality property of Section 3, the overlap of every pair of them is guaranteed to be equivalent (via incidence constraints) to a rotationally symmetric, trivial subgraph. See below for how rotationally symmetric trivial clusters and subgraphs involving incidence constraints are detected and treated. In this case as well, ensuring the first criterion becomes easy. To do so, for each subset of child clusters of C with a nonempty intersection (overlap) subsystem, a DR-sub-plan for the overlap is created. Although there are exponentially many such (necessarily trivial) overlaps in the number of child clusters, the number of distinct overlaps and their total size is bounded by the size of the cluster C, i.e, most of the subsets generate overlaps that are empty or identical to others.

An advantage of using FA to create these DR-plans as well is that they would be created in one pass (as part of a common DR-plan of the entire cluster C) simply by providing an input decomposition whose nodes correspond to these nonempty overlap subsystems. For each cluster in these overlap DR-plans, the assignment of overlap equations should ensure that the first criterion above should hold.

Once these DR-plans are created, since there are atmost $D(m-1)$ overlap equations to be chosen, a standard assignment algorithm - ensuring the 5 above criteria - will take only $O(m)$ time to choose and partition the set of overlap equations.

In the example above, this method will assign 3*2 overlap equations for the point shared by all the clusters, which we denote d: $X_{d,C_1} = X_{d,C_2} = X_{d,C_3}$; $Y_{d,C_1} = Y_{d,C_2} = Y_{d,C_3}$; and $Z_{d,C_1} = Z_{d,C_2} = Z_{d,C_3}$. And 2 overlap equations each for the points a, b and c: $X_{a,C_1} = X_{a,C_2}$; $Y_{a,C_1} = Y_{a,C_2}$; $X_{b,C_2} = X_{b,C_3}$; $Y_{b,C_2} = Y_{b,C_3}$; $X_{c,C_1} = X_{c,C_3}$; and $Y_{c,C_1} = Y_{c,C_3}$.

6.0.4. *Extraneous solutions.* While the set of equations thus obtained for solving the cluster C is guaranteed to be stable and algebraically independent, and their solution set will be finite contain all the valid realizations of C, some of the solutions obtained may not be valid. Consider the following example:

For the 2D example in Figure 26, the overlapped part of C_1 and C_2 consists a, b, c. After ESM performs the above assignment method of overlap equations for C_3, one possible choice of overlap equations could be: $X_{a,C_1} = X_{a,C_2}$; $Y_{b,C_1} = Y_{b,C_2}$; $X_{c,C_1} = X_{c,C_2}$.

Retaining C_1's coordinate system for C_3, the rotation/translation of C_1 is fixed to be the identity hence the left hand sides of the above equations are constants.

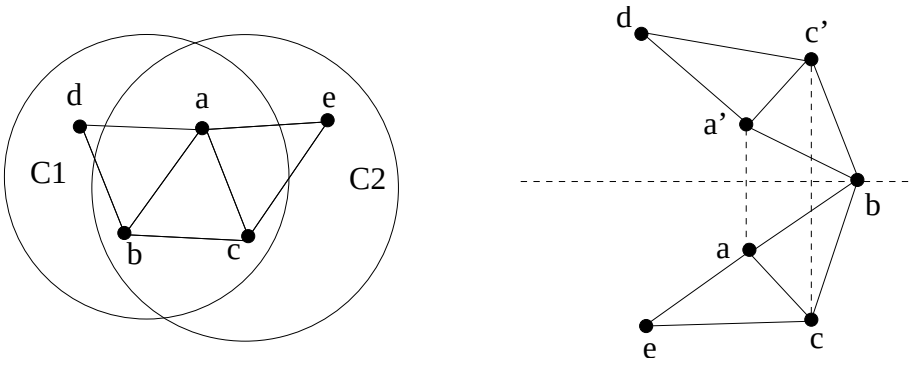

FIGURE 26. A 2D example with extraneous solutions

The right hand sides are expressed in terms of the 4 nonzero rotation/translation variables (2 translations and the sine s and cosine c of the rotation angle) of C_2, which need to be solved for. Setting the fourth equation to be $s^2 + c^2 = 1$, the algebraicSolver is called on the 4 equations in 4 variables. There is only one valid solution for C_3 for any given choice of solutions for C_1 and C_2. However, Two solutions are returned, but in one of them, the locations of a and c in C_1 do not match the locations of a and c in C_2 (although their X coordinate values match).

Hence the various solutions of the current cluster C (assuming fixed solutions for the child clusters) need to be inspected and extraneous solutions discarded.

6.0.5. *Stable algebraic systems in the presence of overconstraints and constraint dependences.* Overconstrained clusters C fall into two types that are handled differently.

The first type of cluster is overconstrained according to the definition in Part I, Section 2 and can be detected using a traditional dof analysis. For such *combinatorially* overconstrained clusters C, overconstraints are first reduced using the method given [**HSY04**]. Inductively, solved child clusters are assumed to be well-constrained. The above method of partitioning overlap constraints for wellconstrained clusters is then applied. Once the solutions are returned by the algebraic-numeric solver, these discarded equations are checked and only those solutions that satisfy them are retained.

The second type of overconstrained cluster is also generically overconstrained but the source of the overconstaint is constraint dependence. See Figures in Section 1 and description in Section 4, explaining the "bananas" problem, and how a good DR-planner marks these as constraint dependences to be "handled with care" during solving. In some of these cases, the left hand sides of these equations are dependent and the right hand sides do not reflect the same dependence. We illustrate with an example how stable systems of equations are built in this case.

For the 3D example in Figure 25(middle, with DR-plan shown on right), the overlapped part of C_1 and C_2 consists of b, c. When ESM builds the algebraic equations for solving C_3, as discussed above, it would obtain 6 equations for b and c in the overlapped part and 1 equation for the distance constraint between a and d: 7 equations but only 6 variables corresponding to the rotation/translation of C_2 into the coordinate system of C_1, chosen as the home cluster, to be the coordinate system

of C_3. In fact, we obtain 10 equations including 3 of the form $s^2 + c^2 = 1$ and 9 variables including the s's and the c's.

To solve the problem, the following modification is made to the method used by ESM. For the vertices in the overlapped part of two child clusters, ESM first checks whether the distances in the local coordinate of each child cluster are consistent (equal within a tolerance). If not, the cluster is returned as inconsistent or unsolvable. If they are, ESM assumes there is one distance constraint between the two vertices in the overlapped part and continues in the usual way, i.e. it only obtains 5 equations for the overlapped part, since it is a rotationally symmetric trivial cluster.

6.1. Incorporating Navigation constraints. Navigation constraints can be used to convert the above ESM methods of user-directed navigation and backtracking 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 off the realizations of the DR-plan's clusters encountered thus far.

Leveraging the modularity of the ESM method discussed above - i.e., its separation from the DR-planner, and its applicability to *any* input DR-plan (generated by any method) - 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. We describe the two common types of navigation constraints below.

One type of constraint used for navigation or picking out desired solutions for any cluster of the DR-plan are Relational or Engineering constraints which typically relate metric parameters of one constraint with that of another without fixing any of them. For example, the sum of the squares of 3 distances (between 3 pairs of points) could be forced to a constant; or the tangent of an angle (between two lines) could be forced to the ratio of some pair of distances (between 2 pairs of points), etc. A method for incorporating these types of constraints into a Shape Recognition (SR) DR-planner (see Section 3) is given in [**FH97**] for triangle decomposable systems).

The other type of common navigation constraints are chirality [**Fud95, FH96b**], also called order type, and related *oriented matroid* constraints. Chirality constraints can be inferred from the input sketch. They describe relative orientations of the objects in the input sketch. For example, for point objects, they 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. Alternatively, a *partial* oriented matroid specification could be specified by the user: these assert intersection or separation of certain pairs of convex hulls of the 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. Depending on their degree, only adhoc approaches are known for dealing with these constraints and these work for any DR-plan. See Section 9.

7. Dealing with inconsistencies and ambiguities

Here we discuss methods for dealing with combinatorially overconstrained and underconstrained systems. Overconstraints caused by constraint dependences such as the "bananas" problem are discussed in Section 4.

7.1. Underconstrained systems. Underconstrained sketches arise often in practice. In fact, as explained in Section 3, the complete set of maximal wellconstrained clusters of underconstrained systems can be read off as the "sources" or "roots" of a FRONTIER's DR-plan [**Sit04b**], [**LS04**]. For underconstrained systems, FRONTIER also provides a sequence of exhaustive lists of constraints(types, not values) that can be generically added (a so-called *completion*) without making the system overconstrained. This was implemented in [**Sit04b**] (2001 version for general 2D constraint systems, 2002 version for a small class of 3D constraint systems, and by the 2003 version for most 3D constraint systems). A proof of why this method works is given below. As pointed out in Part I, Section 1, this should be compared with a method for obtaining completions for the class of triangle decomposable 2D constraint systems which is the subject of the paper [**JASRVMVP03**].

Claim: Let S be the set of root or source clusters of a DR-plan of a 3D or 2D constra int graph G output by FRONTIER. The new constraints that can be added without making any of the clusters (more) overconstrained are exactly those new constraints that relate clusters in S, i.e., those constraints that are not within any of the clusters of S.

Proof. As described in Section 3, and proven in [**LS04**], S consists of a complete set of maximal clusters in G. Clearly adding a new constraint within a cluster of S will cause that cluster to become more overconstrained than before. To prove the converse, assume to the contrary that one of the new constraints causes an overconstraint between two clusters C_1 and C_2 in S. This would imply that either $C_1 \cup C_2$ was a cluster before the addition of the constraint, contradicting the fact that S consists of maximal clusters, or that $C_1 \cup C_2$ contains another wellconstrained cluster D as a subgraph, which overlaps both C_1 and C_2 (atleast one of the overlaps must be on a trivial cluster since $C_1 \cup C_2$ is not a cluster) and D is not contained in either C_1 or C_2. This contradicts the completeness of S.

Once any of these constraints is added, the FA-DR-planner can *update* the DR-plan (as described below) to potentially provide a new list of source vertices, and a new list of constraints between them. This iterative process ends when the DR-plan has a single source vertex, i.e, when the system is wellconstrained. The process gives the sequence of lists required by Part I, Section 3 to obtain a wellconstrained completion of the input underconstrained system.

To obtain the realizable intervals of metric values associated with these constraints is a more difficult problem and is related to the question of describing and navigating the infinite solution space of underconstrained systems (see Section9). However, for the CAD application, there are effective heuristics to infer these values [**FH97, SAK95, Yua99**] from the input sketch of the constraint system.

7.2. Overconstraints. Detection of combinatorially overconstrained clusters is achieved by many DR-planners, including [**LM96**], [**HC01**], and a modification of [**Pab93**] described in [**HLS01a**]. See the message in Figure 7. However,

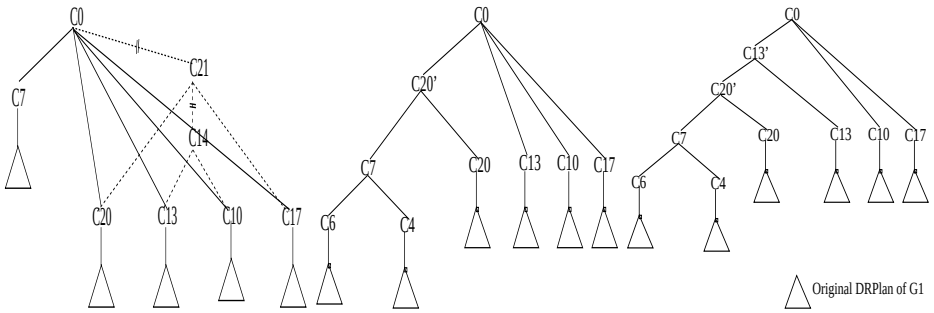

FIGURE 27. Left: unoptimized DR-plan of $G1$ in Figure 11, after reduction of edge $(1, 7)$; modification path is marked; Middle and Right: two optimization steps

efficiently utilizing the DR-plan to isolate the *minimal but complete* set of over-constraints that can be deleted, and efficiently updating the DR-plan after deletion have been addressed only recently in [**HSY04**]. Most solvers ask the user to remove constraints in an over-constrained situation but give poor guidance which constraints to delete. The method in [**HSY04**] is based on showing that there is a unique, well-defined set of *reducible* constraints that can be deleted without making the graph underconstrained. See Figure 27.

The method is further based on key properties of the FA-DR-plan and gives an efficient (generally linear time) solution that retains full generality and moreover:
(i) Traverses the given DR-plan top down to incrementally output this unique set of constraints in reverse solving order, minimizing the need to solve again previously solved portions of the DR-plan;
(ii) Selects constraints from those parts of the constraint system that the user identifies;
(iii) Isolates information that can be routinely stored and maintained as part of the DR-plan, making the above process even more efficient; and
(iv) Automatically *updates* (see Figure 27) the DR plan with optimal reorganization, once one of the reducible constraints is removed, which may cause many lower level clusters of the DR plan to become underconstrained.

This method is described in [**HSY04**] and is part of the FRONTIER solver [**Sit04b**]. This method is purely combinatorial, deceptively simple but makes sophisticated use of the FA DR-plan and hence requires an intricate proof of correctness.

8. Efficient Updates and Online Solving

8.1. Updating a solved constraint system. We discuss the question of efficient updates to the solution of a geometric constraint system (the output in Section 2), given incremental updates to any portion of the input in Section 2.

As it is to be expected, a good DR-plan can be used as a datastructure to systematically realize efficient updates and also to analyze efficiency. Similarly, a good DR-planner (Section 3) and ESM method (Section 5) can be judged by their clarity and efficiency in handling updates.

Before the DR-planning problem was formally defined and the quality and performance of DR-plan structure shown to be crucial for the entire geometric constraint solving process in [**HLS01a**], the question of updates was mostly ignored. Moreover, since the constraint systems are often simple and triangle decomposable, with effectively linear DR-plans whose subsystems were of small size and typically involved solving a single quadratic equation, those constraint solvers such as [**Owe**] that permit updates, implement them by simply redoing the entire system from scratch. For general 3D constraint systems and more complex DR-plans, this approach is intractable.

The properties of FA DR-plans in conjunction with the modularity and clarity of the ESM method described in Section 5 [**SAZK04**], can be used to perform efficient updates [**OSMA01, Sit04b, Sit04a**].

We briefly describe several possible input updates: for each, we list which portions of the output need to be updated and in what possible ways, and sketch methods to do so. In FRONTIER's update mode, changes to the input are allowed at the end of the DR-planning phase, or at any point during the solving and navigation phase of the ESM. See Figure 28.

8.1.1. *Changing Constraint Values.* The first and simplest case is changing the metric value of a constraint e. In this case, no DR-planner is involved. Furthermore, let C be the active cluster where e is resolved (i.e., e is a constraint relating child clusters of C). The only relevant clusters whose realizations will change (for which the algebraicSolver needs to be called again) are those ancestors on the unique path from the active cluster C to a source or root cluster of the DR-plan. In the generic case, there will be no change in the classification of any cluster, or change in the number of realizations of any cluster, or the realization-choices of non-relevant clusters (in order to ensure a solution of the entire system). Hence in the generic case, the update simply requires the ESM to resolve and navigate the realizations of the relevant clusters, keeping all other realization choices fixed.

See Section 4 for a discussion of nongeneric cases and earlier in this section for the underconstrained case where e has no active cluster (it relates 2 source nodes of the DR-plan).

8.1.2. *Adding a Constraint.* Constraints are typically added to convert an underconstrained system into a wellconstrained one. See Section 7.1.

Adding a constraint could require the involvement of the DR-planner. since in this case, the classification of the clusters as well as the entire system could change. I.e, these clusters could now become (combinatorially) generically overconstrained if they were wellconstrained before or the entire system could now become wellconstrained if it was underconstrained before. The former case is immediate to detect. The latter case requires the involvement of the DR-planner. This can be viewed simply as a continuation of FA's DR-planning process from before the addition of e: now e will have to be distributed in the final flow graph containing the source clusters of the DR-plan before the edge was added; and the DR-plan will have to be updated with any new clusters found. For this purpose, the final flow assignments are retained in addition to the cluster structures in the DR-plan, at the end of the DR-planning phase.

Once the DR-planning phase of the update is complete, the ESM's solving phase proceeds as described in the previous subsection on changing constraint value since again, the only clusters affected by the addition of a constraint e are its active and relevant clusters in the new DR-plan.

8.1.3. *Removing a Constraint.* The first case for constraint removal is to convert an overconstrained system into a wellconstrained one. This requires the involvement of the DR-planner and could significantly change the DR-plan and its clusters at every level, since many of the previous clusters may now no longer be wellconstrained. The method of Section 7 [**HSY04**] achieves this with the minimal disruption.

Since the method already deals with previous clusters becoming underconstrained, it directly extends to the second case of constraint removal when the entire wellconstrained system becomes underconstrained.

A third case of constraint removal causes a combinatorially overconstrained or even wellconstrained system to now develop a constraint dependence as in the figures describing the bananas problem in Section 1. However, if no other changes are made, the system continues to have the same set of realizations.

Since constraints have only been removed, all the previous solutions of the the constraint system remain. This is true of any unchanged cluster in the DR-plan. However, constraint removal changes the DR-plan significantly, as described in Section 7, and hence the navigation of the solution space by the ESM would have to be redone atleast for all the new clusters in the DR-plan.

8.1.4. *Adding or Removing a new object or feature.* Addition is handled by putting the new object (and constraints relating it to the existing objects) in the flow graph as an undistributed cluster along with its undistributed incident edges e; and adding the object to the DR-plan as a new source vertex. Now the method reduces to the "Adding a constraint" method discussed above for the undistributed edges e.

Removal is handled by simply removing the constraints that relate the object to the remainder of the constraint system (using the method above).

Adding a new feature to the input decomposition relies on details of the method described in [**SZ04a**], incorporated into the FA DR-planner and can also be found in [**Sit04b, Sit04a**]. Removal of a feature involves direct removal of those clusters - that have now become inessential - from the DR-plan. The process does not involve the DR-planner.

Note: none of these update methods attempt to preserve optimality of the DR-plan, i.e, the sizes of the clusters may not be optimal once the update is complete and is hence not sustainable for large number of updates. See open problem in Section 9.

8.2. Solving a geometric constraint system online. The problem of *online solving* is the following: as the user inputs (sketches) each geometric primitive and constraint of the input constraint system and each feature of the input decomposition, the output - corresponding to the subsystem sketched so far - is immediately available to the user for inspection. This is a difficult question since partially input constraint systems are often underconstrained and realizing those is a a difficult problem. See Sections 7 and 9. The difficulty is however ameliorated by the fact that while the the online solving and partial output process need to be fast and inexpensive, some incorrectness and incompleteness can be tolerated, since the partial output is usually only meant to guide the user and the errors will be rectified when the input is complete and the (correct and complete) offline algorithm is run.

The typical approach to online solving is to run an inexpensive *simplesolver* method that attempts to extend or modify the displayed solution - of the subsystem

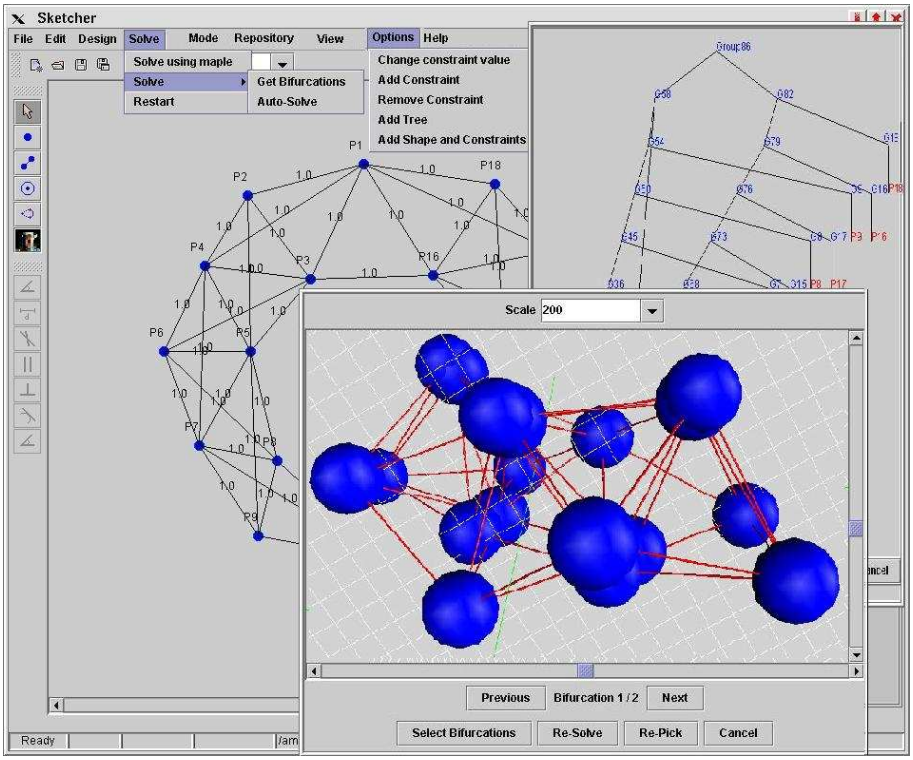

FIGURE 28. Update possibilities shown on pull down menu – updates are possible after the DR-planning stage, during the navigation and after the final solution is displayed

input so far - to include the newly added geometric element or constraint. This method must handle highly underconstrained systems in a simple, lightweight, local fashion, choosing an adhoc extended or modified solution that is as close as possible to the input sketch of the constraint system. If the simplesolver is unable to extend or modify the current solution, then either it outputs the closest approximation it can find, or it then calls the offline algorithm to perform the update and displays the extended solution thus obtained. In the online setting, finding a solution close to the appearance of the input sketch is typically important, and the offline update process must include it as a navigation constraint, see Section 8 and Section 5).

The best performing online simplesolver (for 2D and specific class of "pipe routing" 3D systems), to date is the proprietary commercial D-cubed solver of [**Owe**]. FRONTIER's simplesolver [**OSMA01, Sit04b, Sit04a, JJO04**] works only for 2D and iteratively switches between two methods. Method (i) handles parameterless or nonmetric constraints such as incidence, tangency, etc. as well as angle constraints [**JJO04**] (ignoring distance constraints) and uses nontrivial observations leveraging dependencies of angle constraint graphs. See Figure 29. Method (ii) is a simple, repeated use of the triangle inequality that handles distance (ignoring angles and nonmetric) constraints.

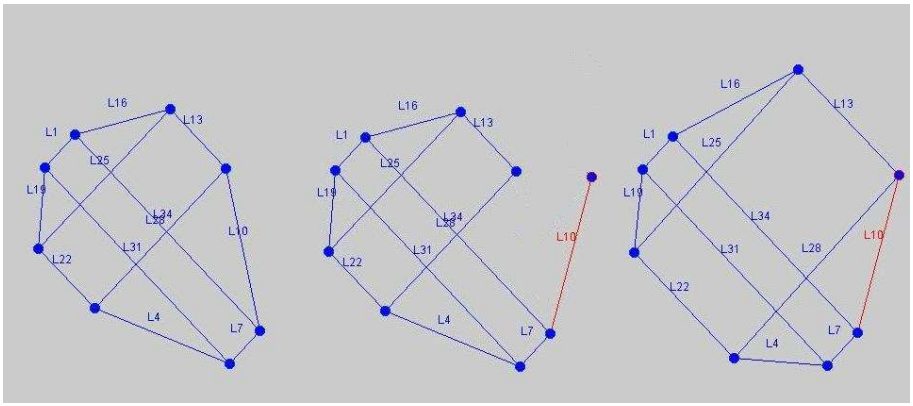

FIGURE 29. Simplesolver that handles angle and incidence constraints online

9. Open Problems, Related areas and techniques

9.0.1. *Inadequacy of combinatorial classification.* This is a longstanding open problem in the *rigidity theory* (see [**GSS93**]) with many conjectures and heuristic approaches but no satisfactory solution; any improvement is desirable: give a completely combinatorial characterization (and associated polynomial time algorithm) for determining if a constraint graph is generically rigid, i.e, wellconstrained, overconstrained or underconstrained. The problem is open not only for 3D systems, but also for 2D systems other than those that contain only points and distances, for example for those that include angle and incidence constraints.

9.0.2. *Underconstrained systems.* This is another longstanding open problem to obtain a satisfactory description and systematic navigation of the solution spaces of underconstrained systems. This is related to the *configuration spaces of generalized 2D and 3D mechanisms* and to *robot arm (inverse) kinematics* with wide ranging applications including newer ones such as *molecular modeling* [**JRKT01, CH88, CNS**] The problem has both combinatorial and algebraic underpinnings. The algebraic part is related to roadmapping real algebraic varieties arising from geometric constraints and also to geometric groups and Lie algebra techniques.

A recent investigation could shed new light on the problem and was kickstarted by a proof of [**CDR03**] of the *Carpenter's rule* conjecture that a polygonal linkage can be convexified without self-intersections. Crucial to the proof is the natural, but clever use of a standard (rigidity based) formalization the concept of *expansive motions*. A constructivization of these expansive motions was provided by [**Str00**], which introduced highly versatile structures called *pseudotriangulations* (wellconstrained completions of 2D polygonal linkage mechanisms which are highly underconstrained systems). These have been used to elucidate several questions [**ARSS03, HOR$^+$03, RSS02, RSS03, Str03**] related to rigidity, topological constraints on configuration spaces of simple 2D mechanisms and promises [**SW**] a combinatorial rigidity approach to 3D robot arm kinematics, molecular modeling etc.

9.0.3. *Inequality constraints and navigation constraints.* The above mentioned problem of describing intervals of realizable distance values - for underconstrained

systems - is related to another longstanding problem of dealing with constraint graphs whose edges are *given* intervals of distance values.

More generally, how to deal with constraint systems which include fixed types of polynomial inequalities? Highly structured polynomial inequalities such as oriented matroid and chirality constraints are used as navigation constraints as mentioned in Section 5 and while that section lists adhoc methods of dealing with such constraints which are effective for limited 2D CAD applications, no formal, provably efficient methods are known: these would be necessary for more complex 3D situations.

9.0.4. *Inverse problem and global constraints.* We consider the *inverse* problem of determining a good, minimal set of local constraints to impose in order to include a desired set of solutions and exclude others. The desired set could have been described by global constraints. This is a crucial problem related to the underconstrained or even wellconstrained solution space navigation problem and is related to the overall issue of using constraints to represent geometric composites.

A typical route is to first decide on which global constraints to impose and then localize them. A classical example is *symmetry constraints*. These are natural in many physical applications as they correspond to minimum energy constraints. These often force solution uniqueness and moreover permit the reduction to a local constraint system. Moreover, symmetry often forces an otherwise underconstrained system, with infinitely many solutions, to become a well-constrained system with few solutions.

The first challenge is in describing the group of symmetries to impose upon the composite and the second is to reduce it into a system of local constraints. These are independently interesting research problems in *computational invariant theory* and *automated geometry theorem proving*, specifically concerning ideal membership and alternative sets of generators.

9.0.5. *Topological and Assembly constraints.* Assembly constraints are of two types: one type explicitly imposes a partial order on the way the constraint system is solved, i.e., an input design decomposition along with a priority order imposed upon its nodes which represent subsystems. As pointed out in Section 3, one of the main advantages of the FRONTIER constraint solver is its ability to generate a DR-plan that incorporates such input decompositions [**SZ04a**] and priority orders. The other type of assembly constraint only implicitly imposes such an order: it concerns intersections that must be avoided during the process of assembly. These are effectively topological constraints on paths in the configuration spaces of the subassemblies. Together with natural assumptions about starting configurations, an assembly (partial) order can be computed.

As mentioned earlier, the primary use of the pseudotriangulations introduced in [**Str00**] is to guarantee expansive infinitesimal motions to deal with topological constraints on paths in the configuration space of an 2D underconstrained system. Generalization to 3D of constructions that guarantee expansive motions would be highly desirable. Initial efforts can be found in [**SW**]

9.0.6. *Optimal DR-planning.* As pointed out in Section 3, the problem of finding an optimal DR-plan - i.e., one that minimizes the size of largest subsystem (minimizes the max fan-in of the DR-DAG) - is NP-complete. No guaranteed (deterministic or randomized) approximation algorithms (or nonapproximability results) are known except for special cases [**LS04**].

9.0.7. *Analyzing Updates and Efficient Online operation.* Detailed methods have been presented in Section 8 to the question of efficiently updating the solution

of a constraint system to reflect incremental changes in the input. The methods use the FA DR-plan as a crucial datastructure. However none of the methods attempt to preserve optimality of the DR-plan. As a result of which, the methods "tolerate" only a limited number of updates: to restore optimality, the DR-plan has to be re-constructed and its clusters resolved from scratch periodically. More sophisticated update methods and (perhaps amortized) analysis is called for.

No efficient online 3D constraint solvers currently exist. They require 2 ingredients: efficient, accurate, offline updates (discussed above); and fast, inexpensive simplesolvers that obtain an approximate solution of a (highly) underconstrained system (special case of a problem discussed below); in particular, they extend an existing solution to an underconstrained system when a constraint or geometric element is added.

9.0.8. *Repeatability and Persistent Naming.* A problem with several of the constraint solvers discussed here is that two different, *nonisomorphic* output DR-plans (and hence the solution space navigation process) could be obtained for the same input constraint system during 2 different sessions, simply because the geometric elements and constraints were input in a different order and considered by the DR-planner in a different order (or perhaps by a different sequence of updates) *Repeatability* is desirable: i.e., to ensure that the DR-planner outputs isomorphic DR-plans DR-plan for isomorphic constraint graphs. Even more desirable would be a systematic manner in which the user could choose one out of several nonisomorphic DR-plans. The ability of a DR-planner to handle an input design decomposition helps, since this restricts the possible output DR-plans to those that incorporate the input design decomposition, but it is not adequate.

One technique to handle this problem is to solve the problem of *persistent naming*: i.e., the constraint solver first renames or renumbers the vertices and edges of the input constraint graph typically based on their local neighborhood information. Regardless of the input ordering of the vertices and edges, the process results in the same renumbering.

References

[AJM93] S. Ait-Aoudia, R. Jegou, and D. Michelucci, *Reduction of constraint systems*, Compugraphics, 1993, pp. 83–92.

[ARSS03] Oswin Aichholzer, Gunter Rote, Bettina Speckmann, and Ileana Streinu, *The zigzag path of a pseudo-triangulation*, Proc. 8th International Workshop on Algorithms and Data Structures (WADS) , Ottawa, Canada, 2003, pp. 377–388.

[BFH+95] W. Bouma, I. Fudos, C. M. Hoffmann, J. Cai, and R. Paige, *A geometric constraint solver*, CAD **27** (1995), 487–501.

[Bru86] B. Bruderlin, *Constructing three-dimensional geometric object defined by constraints*, ACM SIGGRAPH, Chapel Hill, 1986.

[BS03] B. Bettig and A. Shah, *Solution selectors: A user oriented answer to the multiple solution problem in constraint solving*, Journal of Mechanical Design **125** (2003), no. 3, 445–451.

[BVS+93] A. Bjorner, M. Las Vergnas, B. Sturmfels, N. White, and G. Ziegler, *Oriented matroids. encyclopaedia of mathematics. vol. 46. edited by g-c. rota*, Cambridge University Press, 1993.

[CDR03] Robert Connelly, Erik Demaine, and Gunter Rote, *Straightening polygonal arcs and convexifying polygonal cycles*, Discrete and Computational Geometry **30** (2003), 205–239.

[CH88] G. Crippen and T. Havel, *Distance geometry and molecular conformation*, John Wiley & Sons, 1988.

[CLO98] D. Cox, J. Little, and D. O'Shea, *Using algebraic geometry*, Springer-Verlag, 1998.

[CNS] CNS, *http://cns.csb.yale.edu/v1.0/*, NMR structure software.

[Cra79] Henry Crapo, *Structural rigidity*, Structural Topology **1** (1979), 26–45.

[Cra82] Henry Crapo, *The tetrahedral-octahedral truss*, Structural Topology **7** (1982), 52–61.

[FH96a] I. Fudos and C. M. Hoffmann, *Constraint-based parametric conics for CAD*, Computer Aided Design **28** (1996), 91–100.

[FH96b] I. Fudos and C. M. Hoffmann, *Correctness proof of a geometric constraint solver*, J. Comp. Geometry and Applic. **6** (1996), 405–420.

[FH97] I. Fudos and C. M. Hoffmann, *A graph-constructive approach to solving systems of geometric constraints*, ACM Transactions on Graphics **16** (1997), 179–216.

[Fud95] I. Fudos, *Constraint solveing for computer aided design*, Ph.D. thesis, Dept. of Computer Sciences, Purdue University, August 1995.

[GC98a] X. S. Gao and S. C. Chou, *Solving geometric constraint systems. I. a global propagation approach*, CAD **30** (1998), 47–54.

[GC98b] X. S. Gao and S. C. Chou, *Solving geometric constraint systems. II. a symbolic approach and decision of rc-constructibility*, CAD **30** (1998), 115–122.

[GSS93] Jack E. Graver, Brigitte Servatius, and Herman Servatius, *Combinatorial rigidity*, Graduate Studies in Math., AMS, 1993.

[Hav91] T. Havel, *Some examples of the use of distances as coordinates for Euclidean geometry*, J. of Symbolic Computation **11** (1991), 579–594.

[HC01] C. M. Hoffmann and C. S. Chiang, *Variable-radius circles in cluster merging, Part I: translational clusters*, CAD **33** (2001), in press.

[Hen92] B. Hendrickson, *Conditions for unique graph realizations*, SIAM J. Comput. **21** (1992), 65–84.

[HJa97] C. M. Hoffmann and R. Joan-arinyo, *Symbolic constraints in geometric constraint solving*, J. for Symbolic Computation **23** (1997), 287–300.

[HLS97a] C. M. Hoffmann, A. Lomonosov, and M. Sitharam, *Finding solvable subsets of constraint graphs*, Constraint Programming '97 Lecture Notes in Computer Science 1330, G. Smolka Ed., Springer Verlag (Linz, Austria), 1997.

[HLS97b] C. M. Hoffmann, A. Lomonosov, and M. Sitharam, *Finding solvable subsets of constraint graphs*, Springer LNCS 1330 (Smolka G., ed.), 1997, pp. 463–477.

[HLS98a] C. M. Hoffmann, A. Lomonosov, and M. Sitharam, *Geometric constraint decomposition*, Geometric Constraint Solving (Bruderlin and Roller Ed.s, eds.), Springer-Verlag, 1998.

[HLS98b] C. M. Hoffmann, A. Lomonosov, and M. Sitharam, *Geometric constraint decomposition*, Geometric Constr Solving and Appl (Bruderlin B. and Roller D., eds.), 1998, pp. 170–195.

[HLS99] C. M. Hoffmann, A. Lomonosov, and M. Sitharam, *Planning geometric constraint decompositions via graph transformations*, AGTIVE '99 (Graph Transformations with Industrial Relevance), Springer lecture notes, LNCS 1779, eds Nagl, Schurr, Munch, 1999, pp. 309–324.

[HLS01a] C. M. Hoffmann, A. Lomonosov, and M. Sitharam, *Decomposition of geometric constraints systems, part i: performance measures*, Journal of Symbolic Computation **31** (2001), no. 4.

[HLS01b] C. M. Hoffmann, A. Lomonosov, and M. Sitharam, *Decomposition of geometric constraints systems, part ii: new algorithms*, Journal of Symbolic Computation **31** (2001), no. 4.

[HOR+03] Ruth Hass, David Orden, Gunter Rote, Francisco Santos, Brigitte Servatius, Herman Servatius, Diane Souvaine, Ileana Streinu, and Walter Whiteley, *Planar minimally rigid graphs and pseudo-triangulations*, Proc. ACM Symp. Comp. Geometry (SoCG) San Diego, California, 2003, pp. 154–163.

[Hsu96] Ching-Yao Hsu, *Graph-based approach for solving geometric constraint problems*, Ph.D. thesis, University of Utah, Dept. of Comp. Sci., 1996.

[HSY04] C Hoffman, M Sitharam, and B Yuan, *Making constraint solvers more useable: the overconstraint problem*, to appear in CAD (2004).

[HV94] C. M. Hoffmann and P. J. Vermeer, *Geometric constraint solving in R^2 and R^3*, Computing in Euclidean Geometry (D. Z. Du and F. Hwang, eds.), World Scientific Publishing, 1994, second edition.

[HV95] C. M. Hoffmann and R. Vermeer, *Geometric constraint solving in R^2 and R^3*, Computing in Euclidean Geometry (Du D. Z. and Hwang F., eds.), 1995, pp. 266–298.

[HY00] C. M. Hoffmann and B. Yuan, *On spatial constraint solving approaches*, Proc. ADG 2000, ETH Zurich, 2000, p. in press.

[JALS03] R. Joan-Arinyo, M.V. Luzon, and A. Soto, *Genetic algorithms for root multiselection in geometric constraint solving*, Computers and Graphics **27** (2003), no. 1, 51–60.

[JASRVMVP03] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana-Pastó, *Transforming an under-constrained geometric constraint problem into a well-constrained one*, ACM Symposium on Solid Modeling and Applications,Proceedings of the eighth ACM symposium on Solid modeling and applications, 2003, pp. 33–44.

[JJO04] Y. Zhou J. J. Oung, M. Sitharam, *A fast, simple solver for constraints without parameters*, In preparation (2004).

[JRKT01] D. J. Jacobs, A. J. Rader, L. A. Kuhn, and M. F. Thorpe, *Protein flexibility predictions using graph theory*, Proteins: Structure, Function, and Genetics **44** (2001), 150–165.

[Kra92] G. Kramer, *Solving geometric constraint systems: a case study in kinematics*, MIT Press, 1992.

[Lam70] G. Laman, *On graphs and rigidity of plane skeletal structures*, J. Engrg. Math. **4** (1970), 331–340.

[LM96] R. Latham and A. Middleditch, *Connectivity analysis: a tool for processing geometric constraints*, Computer Aided Design **28** (1996), 917–928.

[LS04] A. Lomonosov and M. Sitharam, *Graph algorithms for geometric constraint solving*, submitted, 2004.

[MR97] A. Middleditch and C. Reade, *A kernel for geometric features*, ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications, ACM press, 1997.

[Nel85] G. J. Nelson, *A costraint-based graphics system*, ACM SIGGRAPH, 1985, pp. 235–243.

[OSMA01] J. J. Oung, M. Sitharam, B. Moro, and A. Arbree, *Frontier: fully enabling geometric constraints for feature based design and assembly*, abstract in Proceedings of the ACM Solid Modeling conference, 2001.

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

[Owe91] J. Owen, *Algebraic solution for geometry from dimensional constraints*, ACM Symp. Found. of Solid Modeling (Austin, Tex), 1991, pp. 397–407.

[Owe93] J. Owen, *Constraints on simple geometry in two and three dimensions*, Third SIAM Conference on Geometric Design, SIAM, November 1993, To appear in Int J of Computational Geometry and Applications.

[Pab93] J.A. Pabon, *Modeling method for sorting dependencies among geometric entities*, US States Patent 5,251,290, Oct 1993.

[Rol91] D. Roller, *An approach to computer-aided parametric design*, CAD **23** (1991), 303–324.

[RSS02] Gunter Rote, Francisco Santos, and Ileana Streinu, *On the number of embeddings of minimally rigid graphs*, Proc. ACM Symp. Computational Geometry (SoCG) Barcelona, June 5-7, 2002, 2002, pp. 25–32.

[RSS03] Gunter Rote, Francisco Santos, and Ileana Streinu, *Expansive motions and the polytope of pointed pseudo-triangulations*, Discrete and Computational Geometry - The Goodman-Pollack Festschrift, 2003, pp. 699–736.

[SAK93] N. Sridhar, R. Agrawal, and G. L. Kinzel, *An active occurrence-matrix-based approach to design decomposition*, CAD **25** (1993), 500–512.

[SAK95] N. Sridhar, R. Agrawal, and G. L. Kinzel, *Algorithms for the structural diagnosis and decomposition of sparse, underconstrained design systems*, CAD **28** (1995), 237–249.

[SAZK04] M Sitharam, A Arbree, Y Zhou, and N Kohareswaran, *Solution management and navigation for 3d geometric constraint systems*, submitted, available upon request (2004).

[Sit04a] M Sitharam, *Frontier, an opensource 3d geometric constraint solver: algorithms and architecture*, monograph, in preparation (2004).

[Sit04b] 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*, http://www.cise.ufl.edu/∼sitharam, http://www.gnu.org, 2004.

[Sit04c] M Sitharam, *Graph based geometric constraint solving: problems, progress and directions*, AMS-DIMACS volume on Computer Aided Design (Dutta, Janardhan, and Smid, eds.), 2004.

[SPZ04] M Sitharam, J Peters, and Y Zhou, *Solving minimal, wellconstrained, 3d geometric constraint systems: combinatorial optimization of algebraic complexity*, submitted to ADG 2004, available upon request (2004).

[Str00] Ileana Streinu, *A combinatorial approach to planar non-colliding robot arm motion planning*, Proc. 41st ACM Annual Symposium on Foundations of Computer Science (FOCS), 2000, pp. 443–453.

[Str03] Ileana Streinu, *Combinatorial roadmaps in configuration spaces of simple planar polygons*, Proceedings of the DIMACS Workshop on on Algorithmic and Quantitative Aspects of Real Algebraic Geometry in Mathematics and Computer Science, 2003, pp. 181–206.

[SW] Ileana Streinu and Walter Whiteley, *Single-vertex origami and 3-dimensional expansive motion*, submitted,Dec. 2003.

[SZ04a] M Sitharam and Y Zhou, *Mixing features and variational constraints in 3d*, submitted, available upon request (2004).

[SZ04b] M Sitharam and Y Zhou, *A tractable, approximate, combinatorial 3d rigidity characterization*, submitted to ADG 2004, available upon request (2004).

[Tod89] P. Todd, *A k-tree generalization that characterizes consistency of dimensioned engineering drawings*, SIAM J. Discrete Mathematics **2** (1989), 255–261.

[TW85] T. Tay and W. Whiteley, *Generating isostatic frameworks*, Topologie Structurale **11** (1985), 21–69.

[VLL81] D. C. Gossard V. Lin and R. Light, *Variational geometry in computer-aided design*, ACM SIGGRAPH, 1981, pp. 171–179.

[Whi97] W. Whiteley, *Rigidity and scene analysis*, Handbook of Discrete and Computational Geometry, CRC Press, 1997, pp. 893 –916.

[Yua99] B. Yuan, *Research and implementation of geometric constraint solving technology*, Ph.D. thesis, Dept. of Computer Science and technology, Tsinghua University, China, November 1999.

Current address: Department of Computer and Information Science, University of Florida, Gainesville, Florida 32611

E-mail address: sitharam@cise.ufl.edu