Making Constraint Solvers More Usable: Overconstraint Problem

Christoph M. Hoffmann* Meera Sitharam† Bo Yuan*

December 2, 2002

Abstract

The richness and expressive power of geometric constraints causes unintended ambiguities and inconsistencies during their solution or realization. For example, geometric constraint problems may be turn out to be overconstrained requiring the user to delete one or more of the input constraints, and the solutions must then be dynamically updated. Without proper guidance by the constraint solver, the user must have profound insight into the mathematical nature of constraint systems and understand the internals of the solver algorithm. But a general user is most likely unfamiliar with those problems, so that the required interaction with the constraint solver may well be beyond the user's ability. In this paper, we present strategies and techniques to empower the user to deal effectively with the overconstraint problem while not requiring him or her to become an expert in the mathematics of constraint solving.

We formulate this problem as a series of formal requirements that gel with other essentials of constraint solvers. We then give algorithmic solutions that are both general and efficient (running time typically linear in the number of relevant constraints).

Keywords: Geometric constraint solving, overconstrained problem, redundant constraints, conflicting constraints, inconsistent specifications.

1 Introduction

Product design for manufacture is an activity driven by descriptive information. Increasingly, design includes the explicit inclusion of design constraints into the specifications, especially geometric or geometry-related constraints that impose conditions on the shape of the product. That is, the designer states specific constraints without telling the system in detail how to satisfy them. One goal is to make it convenient for specifying design intent rather than procedure. A second goal is to provide the designer with a succinct, minimal representation of the product which they can specify and edit intuitively. It is then the task of the underlying constraint solver to derive a plan by which to realize and update the constraint representation. i.e, to solve the constraint system and update the solution in response to changes made to the system.

Geometric constraint systems arise in many applications besides CAD, including technical drawing and teaching geometry [39, 23, 38, 30, 19, 21, 22, 36, 2, 4, 16, 17, 10, 11, 12, 13, 14, 18, 9, 15, 7, 8, 27, 28, 3, 29, 24, 1]. Several successful methods have been presented for planning and executing a strategy for solving constraint systems. This is particularly true for solving geometric constraint systems in the plane, although some of the newer approaches including [11, 12, 13, 14, 3, 29], to 3d constraint systems as well.

Informally, a *geometric constraint system* consists of a finite set of geometric objects and a finite set of constraints between them. The geometric objects are drawn from a fixed set of types such as points, lines, circles and conics in the plane, or points, lines, planes, cylinders and spheres in 3 dimensions. The constraints include logical constraints such as incidence, tangency, perpendicularity, etc., and metric constraints such as distance, angle, radius etc. The constraints can usually be written as algebraic equations whose variables are the coordinates of the participating geometric objects.

The solution of a geometric constraint system is a class of valid instantiations of the geometric elements such that all constraints are satisfied. It is understood that a solution is sought in a particular geometry, for

*Purdue University, Work supported in part by ARO Contract 39136-MA, NSF Grant CCR 99-02025, and by the Purdue Visualization Center

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

example the Euclidean plane, the sphere, or Euclidean 3-space. For recent reviews of the extensive literature on geometric constraint solving, see e.g. [13, 20].
Constraint solvers should meet atleast 3 competing challenges.

1. Generality of expression;

2. Efficiency of realization; and

3. Resolution of ambiguities or inconsistencies, and updating (dynamic maintenance).

Note that generality is in a trade-off relationship with efficiency. As the expressive power or generality of the system increases, worst-case performance worsens. Furthermore, multiple solutions may have to be explored, conflicting requirements resolved, redundancies eliminated. Often, these problems overtax the solver and the designer is asked to intervene manually, altering some constraints and dropping others altogether. A given constraint problem may be *overconstrained*, *well-constrained*, or *underconstrained* (formally defined later). Only well-constrained problems are actually solved: under- and overconstrained problems have to be detected somehow and turned into well-constrained problems, upon the designer's intervention. When intervention is required, the constraint solver should offer guidance by presenting viable choices:

- *They should not require mathematical proficiency on the user's part;*

- *They should neither be limited arbitrarily, nor should they include choices irrelevant to the core problem; and*

- *they should be unique in some well defined sense so that the user can expect repeatability of the set of choices presented.*

Efficient Realization Needs DR Plans

A good decomposition of the geometric constraint system is indispensable in dealing with the challenges listed above. Consider the efficiency goal: The cost of solving a geometric constraint system is directly proportional to the size of the largest subsystem that is solved simultaneously with an algebraic or numerical algorithm. This size dictates the practical utility of the constraint solver, since the time complexity is at least exponential in the size of the largest such subsystem. Hence the *optimal* decomposition should minimize the size of the largest such subsystem. Elsewhere, we have analyzed this problem and explained how to find a near-optimal *decomposition-recombination (DR) plan*. Finding a DR-plan can be considered pre-processing: a robust DR-plan would remain unchanged under minor changes to numerical parameters.

DR-plans were used informally by many constraint solvers, but the formal definition of a DR-plan (recalled in Section 2.1) along with performance measures were first given in [13], where an analysis and comparison of various constraint solvers with respect to these measures was also given. Based on these performance criteria, an algorithm, called the Frontier vertex algorithm (FA), was designed in [14]. The FA DR-planner underlies the FRONTIER constraint solver [31, 32, 33, 26] (available as GNU opensource software) [34], which analyzes general 3d constraint systems.

Informally, a geometric constraint solver uses the DR plan to solve problem E by repeatedly applying the following three steps at each iteration i.

1. Find a small solvable subsystem S_i of the (current) system E_i (at the first iteration, this is simply E). This step is indicated by a rectangle in Figure 1.
2. Solve S_i using a direct algebraic or numerical solver.
3. Using the output of the solver, replace S_i by an *abstraction* $T_i(S_i)$, thereby simplifying the system E_i as $T_i(E_i) = E_{i+1}$. This step is indicated by an oval in Figure 1.

An informal requirement on the simplifiers T_i is the following: E_i can be (real algebraically) inferred from E_{i+1}; i.e, we would like every real solution of E_{i+1} to be a solution of E_i as well.

The solver terminates when the small, solvable subsystem S_i found in Step 1 is (a simplified representation of) the entire algebraic system E_i. A DR-plan can be viewed as a directed acyclic graph where each node represents a solvable subsystem S_i (or $T_i(S_i)$, also called a *cluster*) and its descendants represent the different subsystems

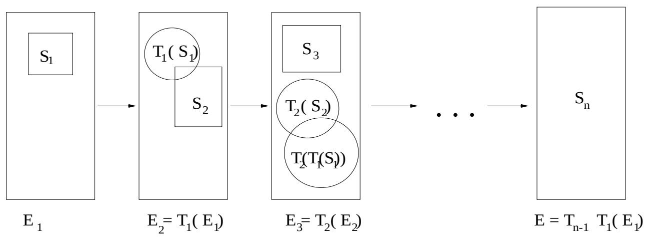


Figure 1: Solving a well-constrained system by decomposition and recombination.

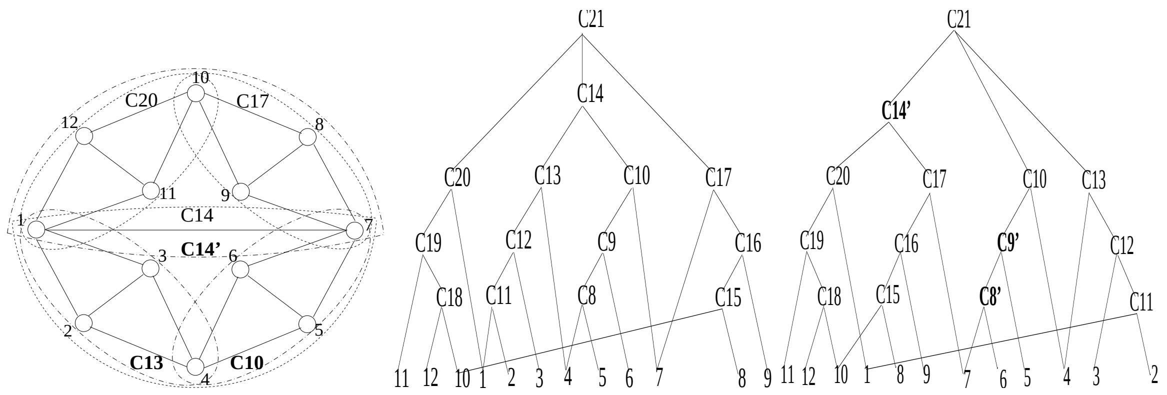


Figure 2: Geometric constraint graph showing well-constrained subgraphs and 2 DR-plans: all vertices have weight 2 and edges weight 1

(found earlier) that were combined to form S_i. Figures 2, 3, 4 (left), 5 (left), 6 (left), all show geometric constraint systems represented as constraint graphs (formally defined in Section 2), and their corresponding DR-plans. If the whole system is underconstrained (as in Figure 6), the solver should still isolate and solve the maximal solvable subsystems.

Inconsistency, Ambiguity and Dynamic Maintenance using DR-plans

Since efficient constraint solvers generally construct a DR-plan, it can and should be used for dealing with ambiguities and inconsistencies with the user's help. This includes how to deal with multiple solutions and how to isolate the under- and overconstrained problem parts. The plan should be a tool to offer the user an incremental list of constraints to add or remove, requiring the least amount of additional work to update the solution and DR-plan.

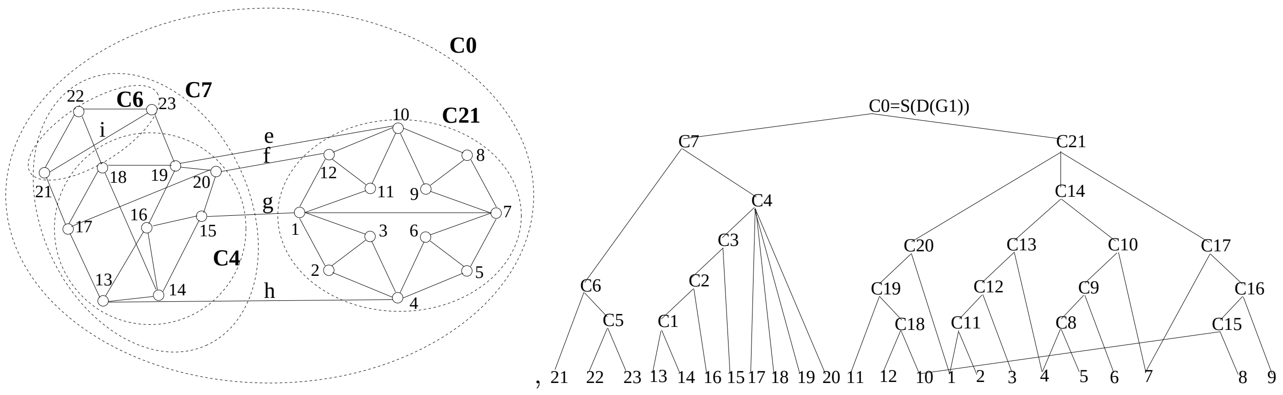


Figure 3: Main example constraint graph G1 and DR-plan all vertices have weight 2 and edges weight 1

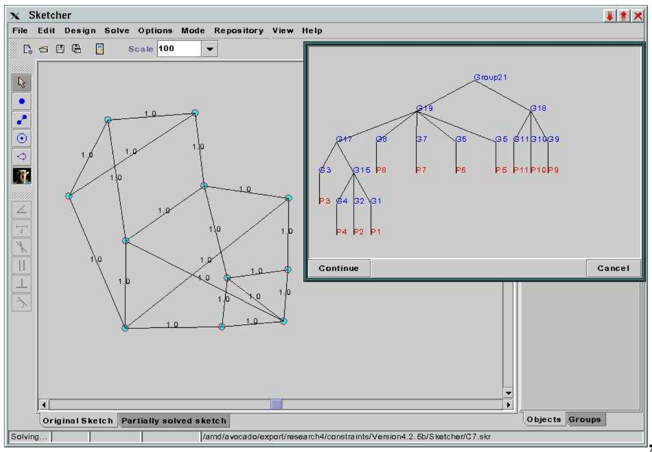
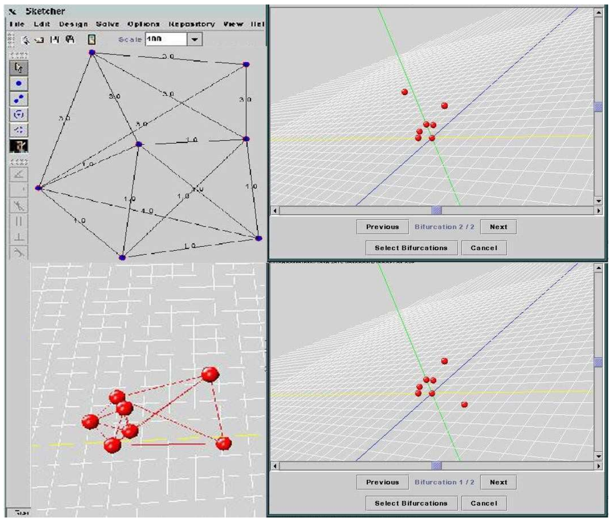

Sketcher
File Edit Design Solve Options Mode Repository View Help
Scale 100
Group21
G19 G18
G17 G8 G7 G6 G5 G11 G10 G9
G3 G15 P8 P7 P6 P5 P11 P10 P9
P3 G4 G2 G1
P4 P2 P1
Continue
Cancel
Original Sketch
Partially solved sketch
Objects
Groups
Solving...
/amd/avocado/export/research4/constraints/Version4.2.5b/Sketcher/C7.skr
,


Figure 4: FRONTIER screenshots. Left: non triangle-decomposable constraint graph and DR-plan: all vertices have weight 2 and edges weight 1. Right: input sketch two 3d solutions and chosen one

The literature has dealt with these issues with varying success. Several heuristics have been proposed for selecting from an exponential, but finite number of solutions of a well-constrained system, [4, 5, 29, 2, 31, 34, 32, 33, 26]. They are based on preserving relative orientation of elements in the input sketch, [2, 29]. These methods can be quite effective and have been used commercially. When the constraint problem changes in a fundamental way, however, a new sketch could be needed. In response, [31, 32, 33] has proposed a new method that works for general decomposition systems and gives the user a visual walk-through of the construction to allow her to isolate the intended solution. See Figures 4, 5. The method scales down efficiently for the special case of triangle decomposable systems.

Underconstrained sketches arise often in practice, so some very good heuristics have evolved for CAD applications. No satisfactory method exists for a systematic navigation of the infinite solution space of underconstrained systems, but many constraint solvers including [21] and [9] detect them. In fact, [14] shows that the maximal well-constrained subsystems of underconstrained systems can be read off as the "sinks" or "roots" of a valid DR-plan. See Figure 6. In many cases, the solver infers heuristically additional constraints from the input sketch of the constraint system, and there are sound ways to do that [6, 37, 40], with input from the user [34, 26, 31, 25, 32, 33]. In addition, the DR plan can be maintained dynamically so that when a constraint system is changed, minimal updates are needed to solve it [34, 26, 32, 33].

Detection of overconstrained clusters is also achieved by existing DR-planners, including [21], [9], and a modification of [24] described in [14]. However, efficiently utilizing the DR-plan to incrementally isolate the *minimal but complete* set of overconstraints that can be deleted, and efficiently updating the DR-plan after deletion have not been addressed adequately so far.

Main Contribution

Current solvers ask the user to remove constraints in an over-constrained situation but give poor guidance which constraints to delete. When a redundant or contradictory constraint is first encountered, the entire cluster is flagged. This identification depends nondeterministically on the DR plan, hence does not penetrate to the core of the problem and is unacceptable in practice.

Investigating overconstrained problems with one redundant or contradictory constraint, we first define precisely what is meant by this term, and how to identify conceptually the *entire minimal subset* of constraints that are in conflict. We show that this subset is unique and well-defined. We offer a simple solution that is general and works in arbitrary dimensions. We then point out the drawbacks: its relative inefficiency, since it ignores the DR-plan whose availablility can be assumed, and its inflexibility. We then give an efficient (generally linear time) solution that retains full generality and moreover

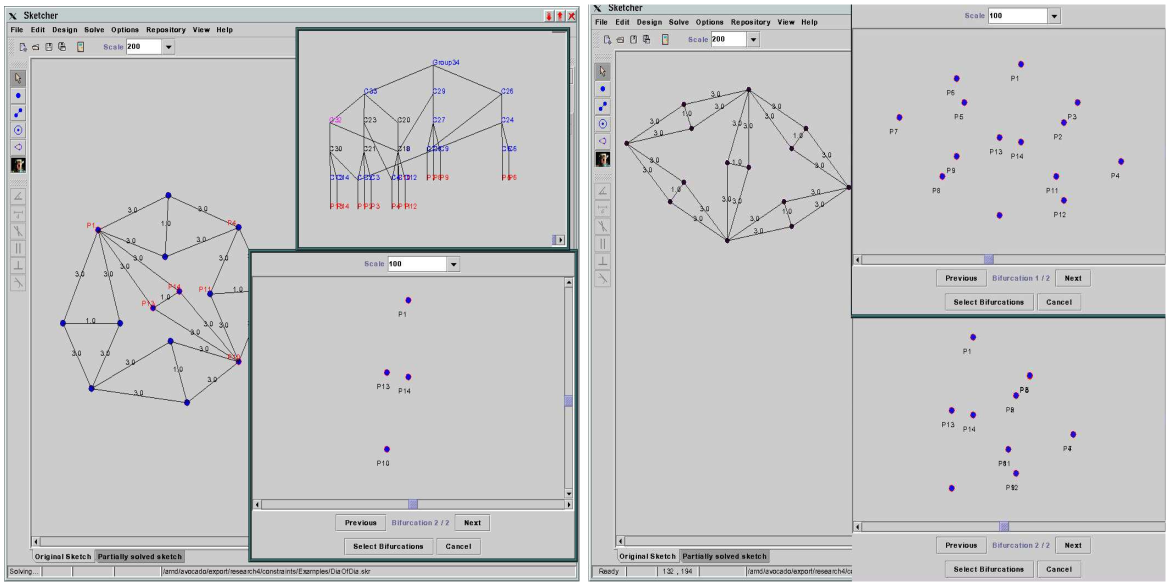
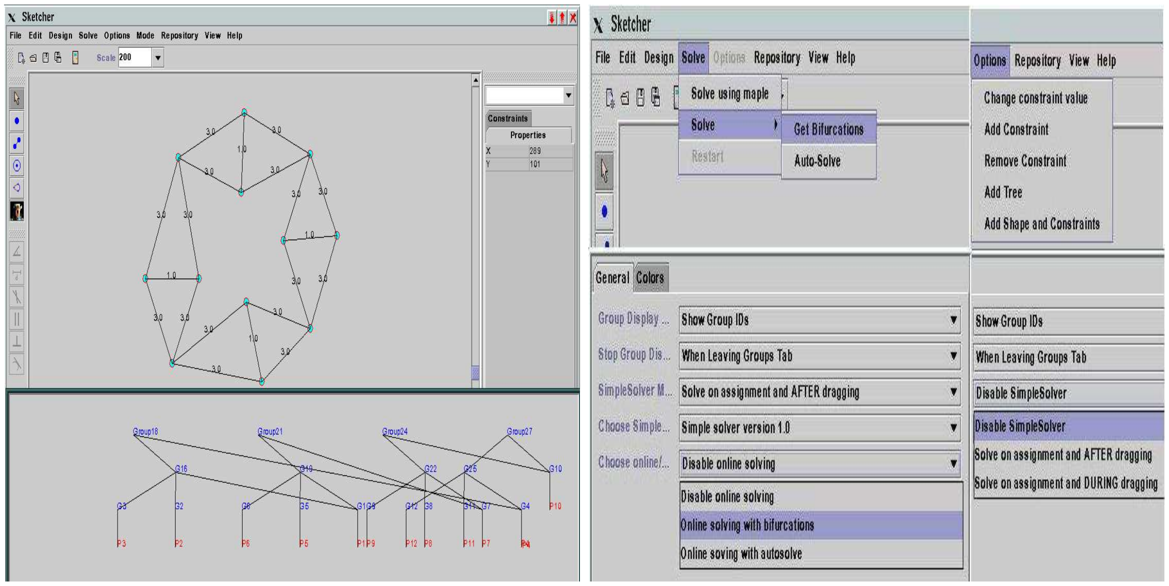


Figure 5: Screenshots from FRONTIER: (Left) 2d sketch, DR-plan and 1 solution possibility for a subsystem; (Right) Various solution possibilities for the entire system, and the final output after choosing subsystem solution on left figure


Sketcher
File Edit Design Solve Options Mode Repository View Help
Scale 200
Constraints
Properties
X 289
Y 101
Group18
Group21
Group24
Group27
File Edit Design Solve Options Repository View Help
Solve using maple
Solve
Restart
Get Bifurcations
Auto-Solve
Options Repository View Help
Change constraint value
Add Constraint
Remove Constraint
Add Tree
Add Shape and Constraints
General
Colors
Show Group IDs
When Leaving Groups Tab
Disable SimpleSolver
Solve on assignment and AFTER dragging
Solve on assignment and DURING dragging
Disable online solving
Online solving with bifurcations
Online soving with autosolve

Group Display ... Show Group IDs
Stop Group Dis... When Leaving Groups Tab
SimpleSolver M... Solve on assignment and AFTER dragging
Choose Simple... Simple solver version 1.0
Choose online/... Disable online solving

Figure 6: (Left) Underconstrained system and DR-plan showing many “roots” or “sinks”, (Right) FRONTIER’s menus – update mode (top right)

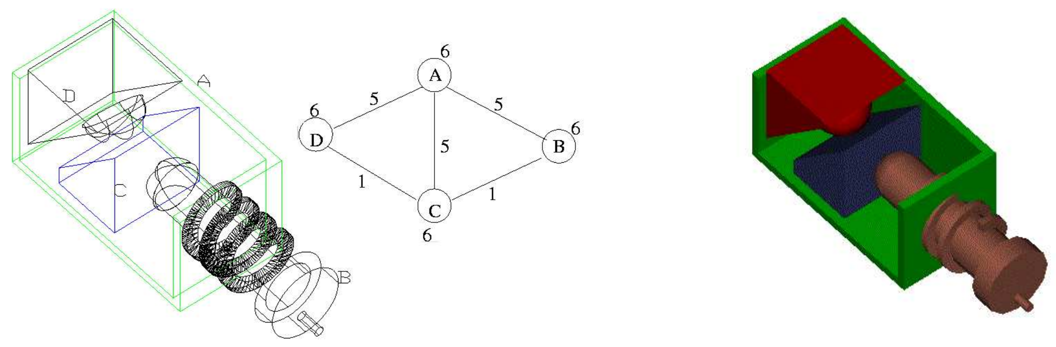


Figure 7: Underconstrained assembly example with 1 extra degree of freedom, and constraint graph

- 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;
- Selects constraints from those parts of the constraint system that the user identifies;
- Isolates information that can be routinely stored and maintained as part of the DR-plan, making the above process even more efficient; and
- Automatically updates the DR plan with minimal reorganization, once one of the offending constraints is removed as chosen by the user.

This algorithm is first described for triangle decomposable systems, and thereafter for general systems. A method is sketched for extending the algorithm *incrementally* to general k-overconstrained graphs. The algorithms have been implemented as part of FRONTIER [31, 32, 33, 26] (available as GNU opensource software) [34].

Organization

In Section 2 we explain geometric constraint graphs and DR-planners and necessary specifics of the Frontier Vertex DR-planner. Section 2 also gives a first formalization of the 1-overconstraintproblem, shows that it is well-posed, gives a network flow-based solution, discusses its drawbacks, and reformulates the problem to account better for the requirements listed before. Section 3 gives an efficient algorithm solving the reformulated problem for a commonly occurring class of 2d constraint systems. Section 4 gives an efficient algorithm for general constraint systems in arbitrary dimensions, including a note on one possible way to extend the algorithm to general k-overconstrained graphs. Conclusions and open issues are indicated in Section 5.

2 Background, Problem Statement, and Initial Solution

Given a constraint problem, a complete DR plan can be constructed without accessing the algebraic solver. A solvable subsystem S_i is located combinatorially using a degree of freedom analysis of the constraint graph, simplified suitably, and the substituted into the larger system E_i, thereby obtaining E_{i+1}. This makes the DR plan generically independent of the valuation of the metric constraints and of the complexity of solving S_i. The formal definition of the DR plan is thus based on viewing the constraint system as the constraint (hyper)graph.

2.1 Constraint Graphs and Solvability

The geometric constraint graph $G = (V, E, w)$ of the constraint problem is a weighted graph with n vertices (representing geometric objects) V and m edges (representing constraints) E; $w(v)$ is the weight of vertex v and $w(e)$ is the weight of edge e, corresponding respectively to the number of degrees of freedom available to object v and number of degrees of freedom (dofs) removed by constraint e. Figures 4 (right) and 7 show 3d examples and their constraint graphs.

In general, the constraint graph is a *hypergraph* with each hyperedge involving any number of vertices. A subgraph $A \subseteq G$ that satisfies

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

is called *dense*, where D is a dimension-dependent constant. Moreover, $d(A) = \sum_{e \in A} w(e) - \sum_{v \in A} w(v)$ is called the *density* of the graph A. The constant D is typically $\binom{d+1}{2}$ where d is the dimension. D represents the *dof* or degrees of freedom associated with the dense graph. In planar Euclidean geometry, we expect $D = 3$, in Euclidean 3-space $D = 6$. If we expect a cluster to be fixed with respect to a global coordinate system, then $D = 0$.

We give some combinatorial properties of constraint graphs that will be shown later to be related to properties of the corresponding constraint systems.

A dense graph with density greater than $-D$ is called *overconstrained.* Note that certain *trivial* overconstrained graphs are common and require special treatment. These are: a single point in 2d or 3d, i.e., graphs with a singleton vertex of weight 2 or 3; a pair of points and a distance constraint between them in 3d, i.e., graphs with 2 vertices of weight 3 and an edge of weight 1 between them. These cases are special because the geometric structures have rotational symmetry. In the following, we exclude larger geometric structures that have rotational symmetry other than the cases listed. A single dense edge of any kind may be considered trivial and treated as a special case for efficiency reasons.

A dense graph all of whose subgraphs have density at most $-D$ is *well-constrained.* A graph G is called *well-overconstrained* if it satisfies the following: G is dense, G has at least one overconstrained subgraph, and remains dense when replacing all overconstrained subgraphs by well-constrained subgraphs. A dense graph is *minimal* if it has no proper, nontrivial, dense subgraph. All minimal dense subgraphs are well-constrained or well-overconstrained, but the converse is not the case. A graph that is neither well-constrained nor well-overconstrained is said to be *underconstrained.*

Fact 2.1 *A dense, nontrivial, underconstrained graph must contain an overconstrained, nontrivial proper subgraph. Conversely, if a graph G of density $-D$ contained a nontrivial subgraph of density $> -D$, then G must be underconstrained. Thus, if a dense G is nonminimal, it may embed a subgraph of density $> -D$ and in that case, G would be underconstrained.*

A generically solvable system always has a well-constrained or well-overconstrained graph, but the converse is not necessarily the case. There are minimal dense graphs whose corresponding systems are generically unsolvable. For a detailed discussion of genericity and the limits of this type of purely combinatorial degree of freedom analysis, see e.g., [14]. Therefore, we restrict ourselves to constraint systems for which well or well-overconstrainedness of the constraint graph implies generic solvability of the constraint system.

2.2 The Frontier Vertex Algorithm (FA-DR planner)

Here we sketch the important features of the FA-DR planner that are necessary for the exposition of the main results. Prior to [13, 14], the DR-planning problem and appropriate performance measures for the planners were not formally defined. Most DR planners were based on decomposing the graph by fixed patterns, such as triangles of 3 points and 3 distance constraints, tri-connectedness etc). This works well in many situations, especially in 2d. Such 2d constraint systems [6, 27, 28, 29] that can be so solved are called *triangle decomposable.* Other methods were more general, and used connectivity and graph matching based approaches [21, 22], [24, 1]. Especially in 3d and also many 2d constraint systems are more complex and these methods do not guarantee high quality, close to optimal decompositions ; see, e.g, Figures 4, 7, and 3. In general, the graph needs to be decomposed into minimal dense subgraphs of arbitrary topology that are as small as possible, their density and minimality the only defining property. An extensive comparison of all of these types of constraint solvers with respect to various formalized performance measures was given in [14]. Our Frontier vertex algorithm (FA) [9], is fully general and optimizes several of the performance criteria defined in [14]. The generality of our algorithm is not entirely reflected in the examples of this paper, which were chosen to facilitate exposition. More general examples can be found at the FRONTIER website [34]. Intuitively, the recursive FA uses the following two steps repeatedly:

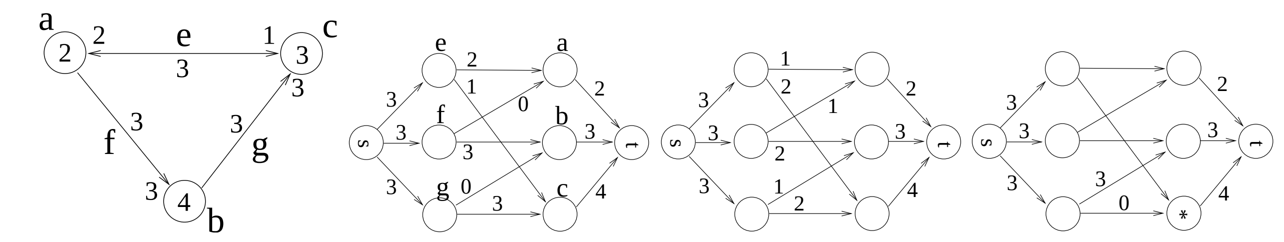


Figure 8: From Left. Constraint graph G with edge weight distribution. A corresponding flow in G^*. Another possible flow. Initial flow assignment that requires redistribution later

1. Find and isolate a minimal dense subgraph (the decomposition step).
2. Simplify transforming the subgraph into the cluster C (the recombination step).

2.2.1 Isolating Clusters: Finding Minimal Dense Subgraphs

The algorithm in [11] that the FRONTIER system employs is a modified incremental network maximum flow algorithm. The idea is to start with the empty subgraph G' of G and add to it one vertex at time. When a vertex v is added, we consider the adjacent edges e incident to G'. For each, 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 vertex v." As illustrated by Figure 8, we may need to redistribute some of the flow again later.

If we are able to distribute all edges, then G' is not dense. If no dense subgraph exists, then the 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, even with redistribution. The last vertex added when this happens can be shown to be in all dense subgraphs $A \subseteq G'$.

Distributing an edge e in G now corresponds to pushing a flow equal to the capacity of (s, e) from s to t in G^*, in the standard bipartite representation of G. This is possible either directly, by a path of the form $\langle s, e, v, t\rangle$ in G^*, or it requires flow redistribution, by the usual method of augmenting paths. If there is an augmenting path, then the resulting flows in G^* provide a distribution of the weight of each edge e in the current subgraph G'. The weight $w(e)$ of each edge e connecting the vertices a and b is split into two parts f_e^a and f_e^b such that $f_e^a + f_e^b = w(e)$ and, for each vertex $v \in G'$, $\sum_{e=(v,*)} f_e^v \leq w(v)$. If there is no augmenting path for the residual flow on (s, e), then a dense subgraph has been found.

Once a dense subgraph G' has been found, a *minimal dense* subgraph inside it can be found by dropping a vertex v and redoing the flow in G'. If a new dense subgraph G'' is so found, then v is dropped and the procedure is repeated. If no such dense subgraph is found, then v must definitely belong to every minimal dense subgraph inside G'. In this way, we can find a minimal dense subgraph in G'.

2.2.2 Simplifying Clusters

Once a minimal dense subgraph S_i is located in G_i, the next step is to simplify it, thereby transforming the constraint graph G_i into a simpler graph G_{i+1}, such that the densities of subgraphs of G_i are preserved in G_{i+1} as much as possible and an optimal DR-plan can be found. It would be simplistic to always condense a cluster to a vertex of G_{i+1}. The cluster 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 the cluster that are not frontier, called the *core vertices*, are contracted into a single vertex. This single vertex is connected to each frontier vertex v of S_i 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.

The process of finding solvable clusters S_i and simplifying them is repeated, until the solvable S_m found is the entire remaining graph G_m. Figure 9 illustrates how FA constructs clusters for the final DR-plan of Figure 3.

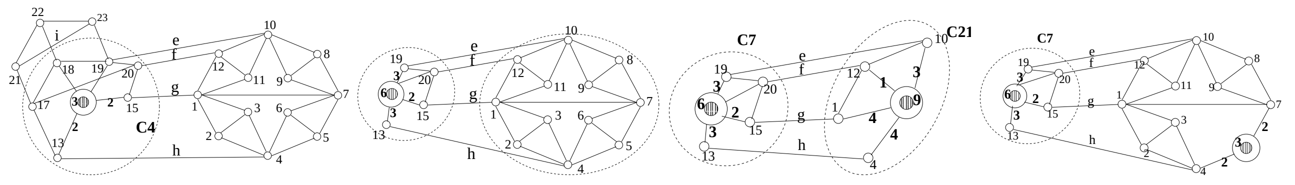


Figure 9: From left: FA's simplification of graph according to DR-plan in Figure 3; clusters are simplified in their numbered order: C4 is simplified before C7 etc.

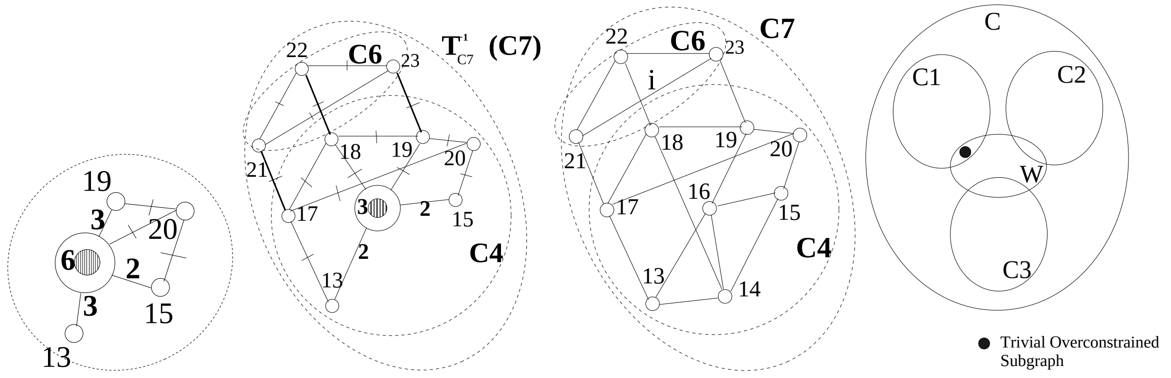


Figure 10: From left. Cluster C_7 of Figure 9. C_7 before transformation: $T_{C_7}^{-1}(C_7)$ – bold edges are relevant edges used for resolving C_7 and both bold and marked edges are unchanged edges. Underlying subgraph G_{C_7} corresponding to C_7 (from graph in Figure 3). Third key property of FA - W must be a child of C

The graph transformation performed by FA is described formally in [14, 15] using *simplifier maps* that provide the vocabulary for proving certain properties of FA and illustrating its superior performance. Moreover, details of the data structures for an implementation have been reported in [32, 33]. There, it is also explained how to deal with special clusters such as trivial, rotationally symmetric, and externally overconstrained clusters.

2.2.3 Key Properties of FA

The following properties of FA are needed for describing and proving correctness of our main algorithm in Section 4. In addition, some key terminology is defined here.

Property 1
The correctness of FA follows from the fact that each internal node in the FA DR-plan represents a well-constrained *cluster* C, obtained in step 2 above, by applying a simplifying transformation T_C to a well-constrained subgraph consisting of child clusters of C. This subgraph is also denoted $T_C^{-1}(C)$ and is isolated as S_i in the decomposition step of some i^{th} iteration of FA. See Figure 10.

If the original graph G is well or well-overconstrained and nontrivial, then the DR-plan is a dag with a single root or sink. If G is underconstrained, then FA finds all of its maximal well or well-overconstrained subgraphs: each sink of the DR-plan represents such a subgraph. The FA DR-plan has the provable property that if the only changes made to G are within a cluster C and these preserve the well-constrainedness of C then they will also preserve the well-constrainedness of the ancestor clusters of C in the DR-plan.

Property 2
The frontier vertices and edges of the clusters C are in 1-1 correspondence with vertices and edges of the original graph G. Furthermore, the edges in the subgraph $T_C^{-1}(C)$ between the child clusters C_i are also in 1-1 correspondence with edges in the original graph G, as illustrated in Figure 10. They represent constraints that are used to resolve or recombine C from the the resolved clusters C_i. They are called the *relevant edges* for C. Conversely, each edge e in G is relevant for a unique cluster C in the DR-plan. The cluster C is called the *relevant cluster* for the edge e. Figure 5 is a FRONTIER screen capture showing candidate solution of the root cluster of the FA DR-plan being recombined or resolved from a chosen solution of child clusters.

Sometimes, as in the graphs of Figure 2 and Figure 5, the constraints between clusters are *implicit*; i.e, they are constraints implied by ***shared objects*** or common frontier vertices between clusters. These are not explicitly listed among the relevant constraints for resolving C. In this case, the only relevant constraints at top level cluster are not explicit constraints but rather implicit shared object constraints.

Property 3
Before simplification, the cluster C is *minimal* (note that cluster minimality is different from the earlier definition of a minimal dense subgraph). I.e., no proper subset of 2 or more child clusters C_i of C induce a dense subgraph of G. Moreover, if the original subgraph G_C underlying C is nontrivial and contains a nontrivial well or well-overconstrained subgraph W, and if the intersection of W with any one of the child clusters C_i of C in $T_C^{-1}(C)$, is a trivial, overconstrained subgraph, then W must itself have been represented as a child cluster C_i in $T_C^{-1}(C_i)$. See Figure 10.

Property 4
Two clusters of the DR-plan do not overlap on non-trivial well-(over)constrained graphs, unless one actually contains the other. The property also holds for the original subgraphs corresponding to the clusters. This property prevents an exponential blow-up of the number of clusters found during the construction of the DR-plan. Accordingly, the total number of clusters in an FA DR-plan is bounded by $O(|V|^d)$, where $|V|$ is the number of vertices of the original graph and d is linear in the dimension of the original geometric space. The depth of the DR-plan is bounded by $|V|$, although in practice the number of clusters is also $O(|V|)$.

The minimal dense subgraph detection needed to isolate each of these clusters could take as many as $O(|V|^2(|V| + |E|))$ steps where $|E|$ is the number of edges of the original graph. In 2d, the overall complexity of the FA DR-planner is bounded by $O(|V|^4(|V| + |E|))$ steps; in practice FA typically takes only $O(|V|^2|E|)$ steps.

Property 5
If G is overconstrained, then for any nontrivial well-overconstrained subgraph W (that may not appear as a cluster in the DR plan $D(G)$), we can read off from $D(G)$ the unique minimal nontrivial cluster C (minimal among those appearing in $D(G)$), whose corresponding subgraph G_C contains W. Other DR-planners besides FA (such as [21] and a modification of [24] described in [14]) also detect such clusters. *But the crucial point to note is that G_C could be significantly larger than W.* It still remainds to efficiently find a minimal well-overconstrained subgraph of W.

In the case of FA DR-plans, uniqueness of this cluster C follows from Property 4. Here, minimality of C means that no descendant cluster contains W. This cluster C is denoted $S(D(G))$ and is the unique minimal cluster C appearing in $D(G)$ where $T_C^{-1}(C)$ is overconstrained. Furthermore, since the child clusters C_i of C are (by the FA simplification) always represented as well-constrained subgraphs in $T_C^{-1}(C)$, the overconstrainedness of $T_C^{-1}(C)$ results entirely from the shared object constraints and the relevant constraints used for resolving C by recombining the C_i.

Property 6
The FA DR-plan respects a hierarchy of well or well-overconstrained features that are specified as part of the the input. That is, these features appear as clusters in the output DR-plan.

2.3 An Initial Problem Statement and Solution

An edge of the constraint graph G is *reducible* if we can reduce its weight by 1 without making G underconstrained. Such edges can be found in well-overconstrained graphs. If G is underconstrained, then a reducible edge belongs to at least one of the maximal well-overconstrained subgraphs of G. By the cut-based definition of underconstrained graphs, the reducible edges do not belong to any minimal cut of weight D in G unless the cut is trivial. An overconstrained graph is *1-overconstrained*, if it, or each of its maximal well-overconstrained subgraphs, becomes well-constrained as soon as one reducible edge has its weight reduced by 1. In the following, we assume that 1-overconstrained graphs are nontrivial. We state formally the problem addressed in this paper and, based on the basic constraint graph properties, we give an initial solution.

Problem: Give an efficient algorithm that takes as input a 1-overconstrained graph G, and outputs its set of reducible edges, $L(G)$.

Fact 2.2

(i) *Any nontrivial 1-overconstrained graph G may have many (1-)overconstrained subgraphs, but it has a unique minimal, nontrivial, 1-overconstrained subgraph U. That is, no proper subgraph of U is 1-overconstrained.*

(ii) *An edge is in U if and only if every (1-)overconstrained subgraph of G contains it.*

(iii) *U is 1-well-overconstrained.*

(iv) *U consists of exactly the reducible edges of G.*

Proof: (i) Suppose U is not unique and there are two distinct minimal, nontrivial 1-overconstrained subgraphs U_1 and U_2. Then either their intersection is 1-overconstrained, or their union has density at least $D + 2$, contradicting the 1-overconstrainedness of G. If their intersection is 1-overconstrained, then U_1 and U_2 cannot be minimal. Statement (ii) follows from (i) and from minimality and uniqueness of U. Statement (iii) follows from U's minimality and by Fact 2.1.

For (iv), assume that the minimal 1-overconstrained graph U is not $L(G)$. First, assume that one of the edges e outside U is reducible and reduce its weight by 1 obtaining G'. Then G' still has density $-D$, but it has a subgraph U of density $-D+1$, and, by Fact 2.1, G' must be underconstrained, contradicting the assumption that e was reducible. Conversely, assume that one of the edges in U is not reducible. Then reducing its weight by 1, the new graph G' must be underconstrained, but it is still dense. In G', however, U and all other 1-overconstrained subgraphs of G are no longer 1-overconstrained, since they all contain U. This contradicts Fact 2.1. □

Initial Algorithmic Solution

Using Fact 2.2, we can immediately obtain an $O(|V|^2(|V|+|E|))$ network flow based algorithm for our problem, by adapting the minimal dense subgraph location algorithm of Section 2.2.1. This works for completely general geometric constraint hypergraphs in arbitrary dimensions. Simply locate a minimal subgraph of density $-D+1$ by first finding a subgraph of density at least $-D+1$, distributing an additional flow of $D+2$ beyond the weight of the edge. Once this graph is found, locate a minimal 1-overconstrained subgraph U in it by deleting vertices one by one, so finding a subgraph of density $-D+1$ within the resulting subgraph. The subgraph U so found must be $L(G)$ by Fact 2.2.

2.4 Drawbacks and Modified Problem Statement

The problem statement and solution above are unsatisfactory since they do not satisfy the requirements discussed informally in the Introduction:

Requirement 1:

Given a DR-plan $D(G)$ as input, the algorithm should be significantly more efficient (preferably linear time) than the one given above, by utilizing an existing DR-plan to find the reducible edges of G. (The above algorithm ignores any DR-plan information).

Requirement 2:

The list of reducible edges should be compiled *incrementally* by traversing down the DR-plan, cluster by cluster, starting from the unique overconstrained cluster $S(D(G))$ as described in Property 5 of FA.

Upon examining $S(D(G))$, the first part of the edge list will consist of the reducible edges in G that are unchanged edges of $S(D(G)$. That is, these edges are relevant in recombining already solved child clusters of $S(D(G))$. The next sublists output are the reducible edges in G that are relevant to the child clusters of $S(D(G))$ and so on.

In general, if a cluster A in $D(G)$ is an ancestor of a cluster B then the sublist for A will be output before the sublist for B. *Thus, the edges or constraints in $L(G)$ are output in the reverse order in which they would be solved.*

Requirement 3:

The sublist of reducible edges for a particular cluster C should be output on demand, preferably in a manner that inspects exclusively the clusters on the path in the DR-plan from the cluster $S(D(G))$ that contains C, upto the cluster C.

This requirement is meaningful for instance in the case of FA DR-plans, because those plans reflect a designer's conceptual feature hierarchy.

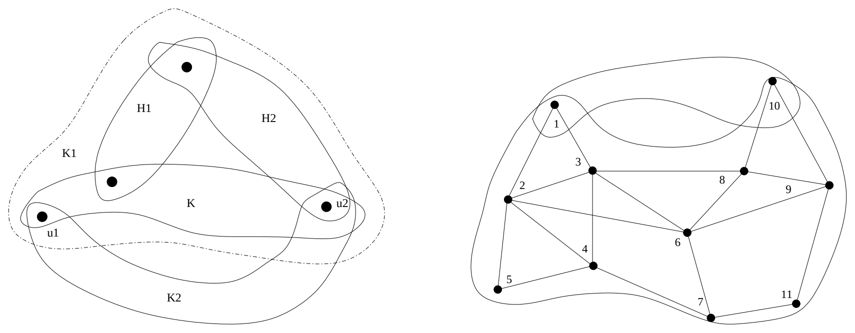


Figure 11: Left: Clusters H_1 and H_2 can be dropped from consideration. Right: Cluster pruning alone does not necessarily yield the critical subproblem.

Requirement 4:
The DR-plan should be updated efficiently when a reducible edges is reduced. The update should inspect only clusters that are actually changed by the reduction.

In the next section, we concentrate mostly on Requirements 1, 2 and 3. Requirement 4 follows from our algorithmic solution with some routine work, and we only discuss it informally.

3 Triangle Decomposable 2d Constraint Systems

For most 2d constraint systems arising in applications, it is sufficient to use the triangle decomposition algorithm of [6] whose clusters are formed by combining three child clusters pairwise sharing one geometric element. When points and lines are the geometric vocabulary, all vertices have weight 2 and edges have weight 1; hence 1 well-overconstrained graphs have a number of edges $|E|$ that is linear in the number of vertices $|V|$. Elementary clusters consist of two vertices and an edge between them. As shown in [6], overconstrained problems are detected by two clusters sharing more than one geometric element, so that the 1 well-overconstrained case implies two clusters that share two geometric elements. This includes adding an extra weight-1 constraint into the cluster, since such a constraint, along with two incident weight-2 vertices, can be considered a cluster. We explain how to obtain the the set of reducible edges. Note that the edge weight 1 implies that a reducible edge is reduced by removing it.

Let K_1 and K_2 be the overlapping clusters, u_1 and u_2 the shared vertices between them. Our algorithm first examines the decomposition tree to find cluster merges in K_1 and K_2 in which both u_1 and u_2 are in the same cluster, as illustrated in Figure 11 (left). It is clear that if the clusters H_1 and H_2 can be removed, then the resulting, smaller cluster remains 1-overconstrained. Moreover, since both H_1 and H_2 must be wellconstrained, deleting any constraint within those clusters cannot change the original problem into a well-constrained one. Thus, iteration of the pruning step obtains a cluster K' that must contain the unique, minimal 1-overconstrained subgraph and hence all of the reducible edges by Fact 2.2.
As noted in [13, 14], constraint-graph decomposition is not deterministic, but it satisfies the Church-Rosser property. Therefore, cluster pruning, in general, does not necessarily preserve the set of reducible edges, and an example is shown in Figure 11 (right). What is needed is a tree re-ordering that exhibits the remaining redundancies. The decomposition method of cutting [40] achieves this, as follows.

Let $G = (V, E)$ be the constraint graph known to be 1-overconstrained. The *cutting step* finds a vertex $v \in V$ such that the weight $w(v)$ is equal to the sum of the weights of the incident edges. Such a vertex can be constructed as the last step of the solution plan, and can therefore be removed from G without changing whether G is structurally over-, well-, or underconstrained. Cluster pruning can be considered a generalized cutting step. We thus obtain the following algorithm for identifying the critical subproblem:

1. Repeat cluster pruning until no further clusters can be removed.

2. Repeat the cutting step until no further vertices or subclusters can be removed.

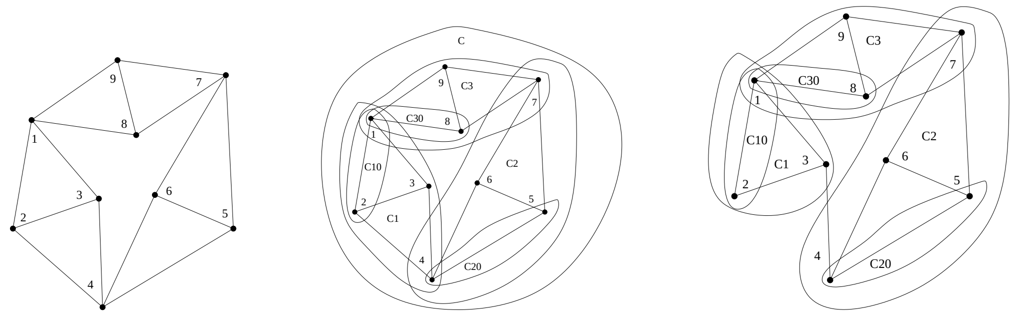


Figure 12: Left: A constraint graph Middle: its decomposition structure; Right: The result of edge reduction

The algorithm, as stated, has quadratic complexity. This can be lowered to $O(|E|\log(|E|)) = O(|V|\log(|V|))$ using a priority queue. However, we can achieve a linear-time algorithm.

Consider the cluster and mark the two shared points. All other vertices are unmarked. Then, consider the vertices in the reverse order in which they were added to the cluster. We will accumulate the set U of reducible constraint edges. U is initially empty. Perform the following for each vertex v encountered:

1. If the vertex v is not marked, delete it and delete the constraints by which it was added.
2. If v is marked, then
3. If there are other vertices in the remaining cluster that are marked, then include into U the edges of the constraints by which v was added and mark the vertices incident to the constraint edges.
4. Otherwise, if v is the only marked vertex, terminate the process.

It is intuitively clear that the marked edges are precisely the reducible constraints that contribute to the overconstrained situation. Using reference counts, we can implement the test of remaining marked vertices in constant time, so that the algorithm overall is linear-time.

Example. Consider the cluster shown in Figure 11(Left). Vertices 1 and 10 reveal the 1-overconstrained situation when merging the lower and the upper cluster. Assume that the lower cluster has been constructed by choosing the edge $(1, 2)$ as cluster core and sequentially extending the cluster in the order of vertex enumeration.

Initially, vertices 1 and 10 are marked, and in the lower cluster the vertices are examined in reverse numerical order. Vertex 11 is examined first. Since it is not marked, it is deleted along with the edges (7,11) and (9,11). When examining vertex 10, we add the constraint edges (8,10) and (9,10) to U and mark vertices 8 and 9. Examining 9 next, we add edges (8,9) and (6,9) to U and mark 6. On examining 8, we add (3,8) and (6,8) and mark 3. Vertex 7 is unmarked and is deleted along with edges (4,7) and (6,7). Eventually, the process ends with a set

$$L = \{(8,10), (9,10), (8,9), (6,9), (3,8), (6,8), (3,6), (2,6), (1,2), (1,3)\}$$

Note that the vertices $11, 7, 4, 5$ are unmarked. □

The algorithm is straightforwardly extended to cluster pruning. Here, we have to maintain a mark reference counter for the clusters themselves, in addition to the intra-cluster reduction explained before. Therefore, the critical constraint set can be found in linear time.

Updating Triangle-Based DR-Plans: Constraint Removal

Removing a constraint corresponds to removing an edge from the constraint graph. To minimize the changes of the decomposition, we only modify or destroy clusters that are based on this edge; that is, clusters whose induced subgraph includes the edge. Figure 12 (left) is a constraint graph G and Figure 12 (middle) illustrates the decomposition structure of the G. For example, if the edge $(2, 4)$ is removed, the cluster C is destroyed and the cluster C_1 is modified, but other clusters remain unchanged. Figure 12 (right) shows the result of the removing operation.

Note: One difficulty of this algorithm arises from the fact that removal of a constraint may not preserve or retain triangle decomposability of the resulting graph. In the case of solvers that rely on triangle decomposability, therefore, we identify additionally the subset of constraints whose removal retains triangle decomposability. This can be done by deleting each constraint in turn and testing decomposability. Since 1-overconstrained problems are identifiable early, and in view of our experience that the critical subproblems of such overconstrained problems are usually not very large, this approach is efficient.

4 The Algorithm for general DR plans

We now describe the algorithm Overlist, which takes as input a general geometric constraint hypergraph G that may not be triangle decomposable and is known to be 1-overconstrained. Therefore, as in the previous section, $|E| = O(|V|)$. The algorithm uses FA to generate a DR plan $D(G)$ and and outputs the complete list $L(G)$ of reducible edges in G incrementally with respect to $D(G)$, and in such a way as to satisfy all of the requirements of Section 2.4. First, we present the core algorithms. Adaptations of the algorithm are briefly sketched in Section 4.1, followed by illustrative examples in Section 4.2. Correctness proof and the complexity analysis follow.

Algorithm Overlist uses a DR-plan $D(G)$ and the unique smallest nontrivial 1-well-overconstrained subgraph corresponding to a cluster $S(D(G))$ in $D(G)$. *Recall that the subgraph corresponding to $S(D(G))$ may be significantly larger than the unique minimal 1-well-overconstrained subgraph of G which, in general, need not appear as a cluster in $D(G)$.* This is a limitation not only of the FA DR-planner, but other DR-planners such as [21] and [24] that in some form or another detect overconstrained clusters. This is the starting point of our main contribution.

The tuple $(G, D(G), S(D(G)))$ is input to the algorithm Planover, the main technical contribution of this section. Algorithm Planover further calls 2 recursive subroutines Unravel and Unravel* that incrementally output the list $L(G)$ while traversing the sub-DR-plan of $D(G)$ rooted at the cluster-node $S(D(G))$. See Figure 17 for the overall control flow of these algorithms. While the main contribution here is the Algorithm Planover which takes the FA DR-plan as input, the text in Figure 17 also illustrates the importance of the FA DR-plan's key properties from Section 2.2.3, by using the example graphs $G2, G3, G4, G5$ and $G6$ (Figures 13, 14 15 and 16), which are variants of the example graph $G1$ in Figure 3.

```
ALGORITHM OVERLIST(G)
Step 1: Run the FA DR-planner to get the DR-plan $D(G)$ and the cluster $S(D(G))$ representing the unique
smallest 1-well-overconstrained subgraph of $G$ that appears as a cluster in $D(G)$. Since $G$ is known to be
1-overconstrained, we know that this subgraph is nontrivial.
Step 2: PLANOVER$(G, D(G), S(D(G))$

ALGORITHM PLANOVER$(G, D(G), S(D(G))$
$L(G) := \emptyset$; $C := S(D(G))$
UNRAVEL*$(C, \emptyset)$; Output $L(G)$

ALGORITHM UNRAVEL*$(C, P)$
$L_C :=$ set of unchanged edges (from $G$) in $T_C^{-1}(C)$ between the child clusters $C_i$ of $C$;

$L(G) = L(G) \cup L_C$; Output $L_C$
$P_C :=$ set of contact points, i.e, those vertices of the child clusters $C_i$ in $T_C^{-1}(C)$ that are (a) either present
in more than one $C_i$ (these represent "shared-object" or "incidence" constraints); or (b) participants in some
edge(constraint) in $L_C$. Note that these contact points are necessarily Frontier vertices of the $C_i$'s and are hence
unchanged vertices from $G$)
$P_C^* = P_C \cup P$
For each child cluster $C_i$ of $C$ in $D(G)$,
UNRAVEL$(C_i, P_C^* \cap C_i)$

ALGORITHM UNRAVEL$(C, P)$
If $C$ represents a singleton vertex of $G$, then return NULL
Else,
    If all the points in $P$ lie inside a single child cluster $C_i$ of $C$ in $T_C^{-1}(C)$
```

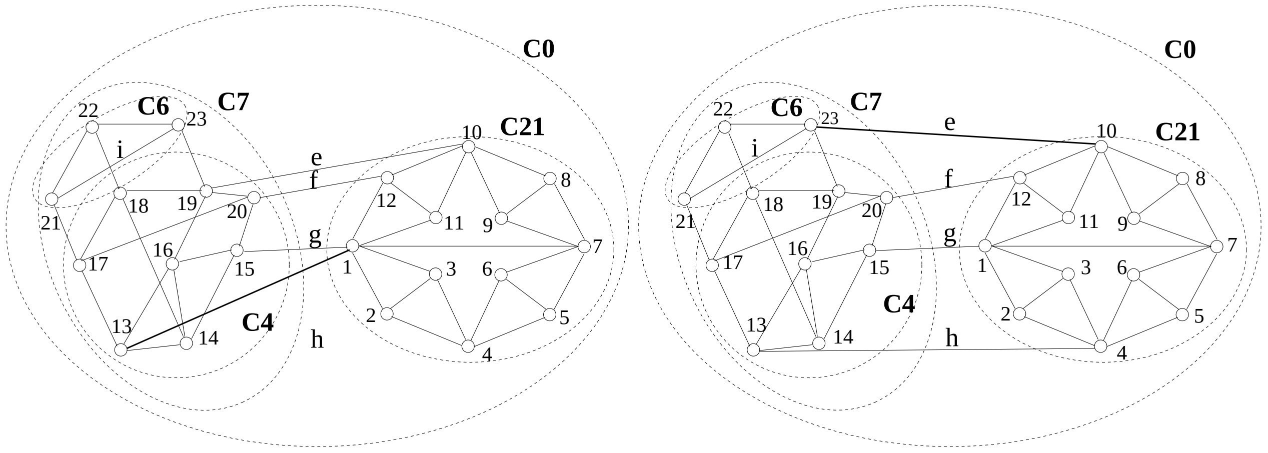


Figure 13: Left - Graph $G2$: edge h changes from $G1$; Right - Graph $G3$ edge e changes from $G1$; in both cases, no change in FA DR-plan

then UNRAVEL(C_i, P) (if there is more than 1 such child cluster C_i, pick *any one*)

Else UNRAVEL*(C, P)

Notes on the pseudocode:
– Algorithm OVERLIST is never called on clusters C corresponding to trivial well-overconstrained subgraphs of G, since $S(D(G))$ corresponds to a nontrivial subgraph, by the results in Sections 2.2.
– In Algorithm PLANOVER, the complete list of edges $L(G)$ is output at the end, however, the list is also incrementally output when sublists are encountered at clusters of $D(G)$ during UNRAVEL* – as explained below.
– UNRAVEL*, is never called on clusters C corresponding to singleton vertices of G; hence both L_C and P_C are well-defined.
– In the description of UNRAVEL*, the C_i are obtained from $D(G)$ – see Section 2.2; also, L_C could be empty if C resulted from only implicit or "shared-object" constraints between its child clusters.
– In the description of UNRAVEL*, L_C is output at this stage as the incremental sublist of $L(G)$ obtained from C. By Section 2.2.3 the edges in $L(C)$ represent the relevant explicit constraints used to recombine or solve C from the already solved child clusters C_i. Thus it immediately follows that the Requirement 2 of the problem statement in Section 2.4 is met by this algorithm
– In the description of UNRAVEL*, $P_C^* \cap C_i$ represents those contact points in C_i that are inherited from C.
– In the description of UNRAVEL, the call UNRAVEL(C_i, P) is well defined for the following reason: the points in P are always vertices from G that are unchanged in C, they are not only frontier vertices in C but also well-defined as frontier vertices of C's children C_i in $T_C^{-1}(C)$

4.1 Some Adaptations of the Algorithm

A key property of our algorithm is that with small adaptations, it can easily and efficiently answer the following types of user queries; see Requirement 3 of the problem statement in 2.4): "Is this edge e reducible?" or "Given a particular cluster C of the given DR-plan $D(G)$, show me the list L_C of reducible edges relevant to C, if it exists."

Observation 4.1 *The Algorithm* UNRAVEL *is called only on a subtree of $D(G)$. In fact, it is a subtree of the subplan of $D(G)$ that is rooted at $S(D(G))$. The sublist L_C is formed only on those vertices of this subtree where* UNRAVEL* *is called.*

This observation is illustrated in Section 4.2 and can be used as follows. For an edge-reducibility query, locate the unique minimal relevant cluster C for e. Then, for both types of queries, the algorithm can give immediately an efficient solution that requires only looking at the path of clusters from $S(D(G))$ to C in the DR-plan $D(G)$. This path is called the *unravelling path*: check whether UNRAVEL gets called on all of these

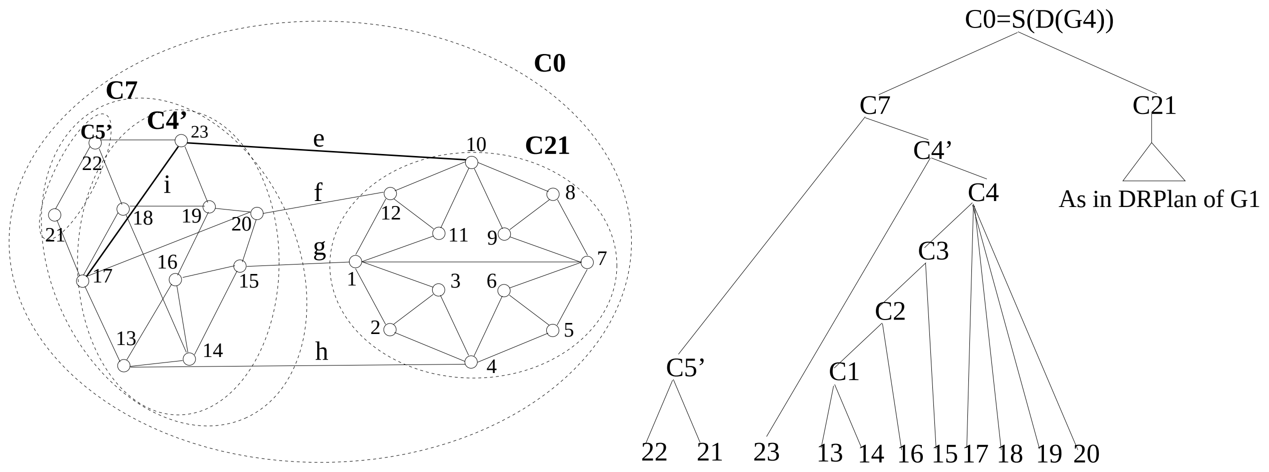


Figure 14: Graph $G4$: edge e and i change from $G1$; Right: changed DR-plan

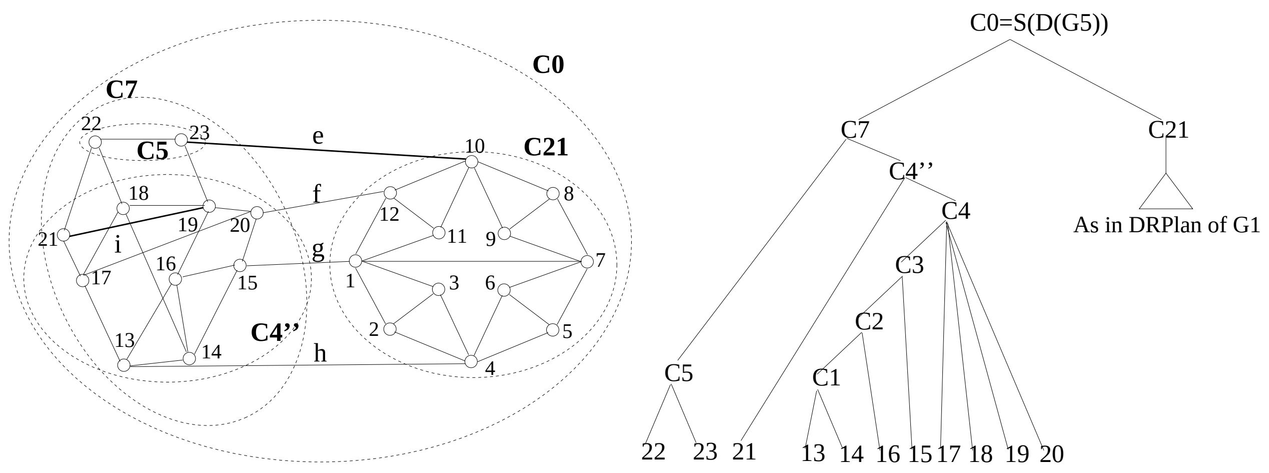


Figure 15: Graph $G5$: edge e and i change from $G1$; Right: changed DR-plan

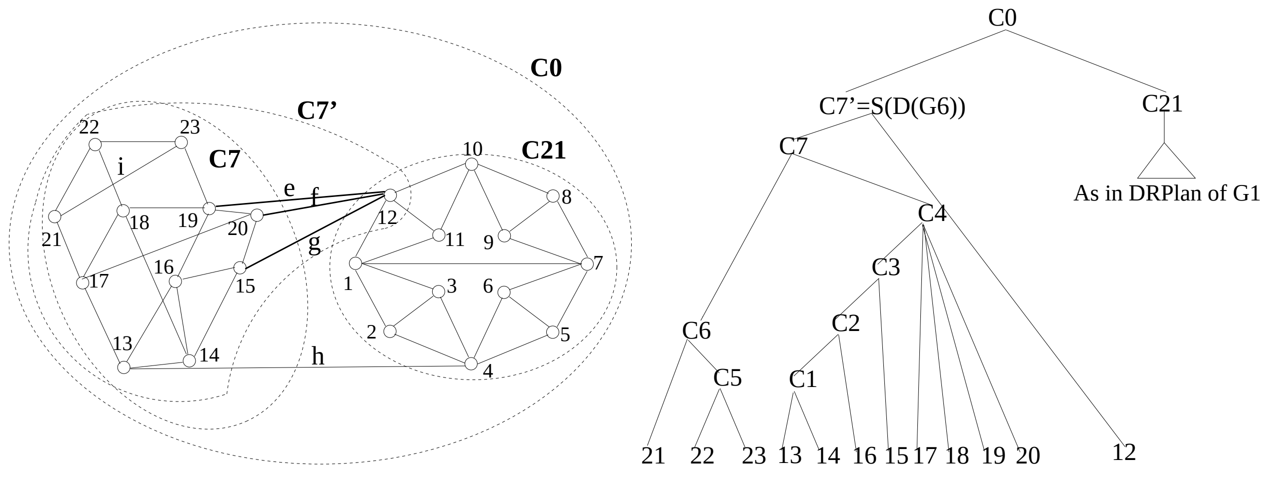
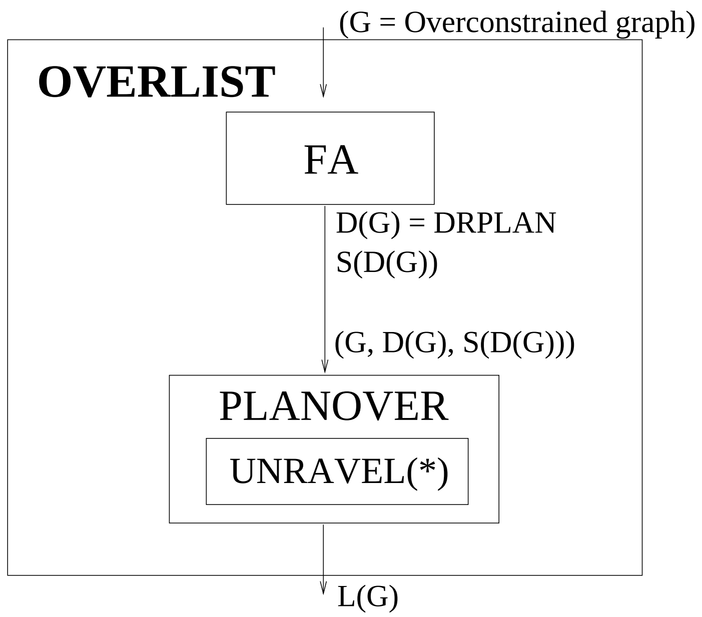


Figure 16: Graph $G6$: edge f and g changed from $G1$: Right: changed DR-plan as well as input cluster $S(D(G))$ for Planover



Case of G2 & G3:
* D(G2) & D(G3) are identical to D(G1)
* S(D(G2)) & S(D(G3)) are also identical to S(D(G1))
* L(G2) & L(G3) .i.e. output of PLANOVER changes since G2 & G3 are different from G1

Case of G4 & G5:
* D(G4) & D(G5) both change from D(G1)
* S(D(G4)) & S(D(G5)) are unchanged
* Change in L(G4) & L(G5) .i.e. output of PLANOVER crucially depends on change in D(G4) & D(G5)

Case of G6:
* S(D(G6)) changes from S(D(G1))
* Change in L(G6) .i.e. output of PLANOVER crucially depends on this change

Cases G4,G5,G6 when compared to case G1 illustrate that correctness of OVERLIST depends not only on PLANOVER but also on the properties of FA's output (.i.e. entire input to OVERLIST).

Figure 17: Overall control flow of algorithms; Text: differences in the example variants $G1$, $G2$, $G3$, $G4$ of Figure 3, and Figures 13, 14, 15, 16
.

clusters, starting from the top of the path and following the contact points. The rest of the graph can be ignored. Finally, check whether UNRAVEL* gets called at C. If so, then the relevant edges of C are reducible edges that would be output as L_C by Algorithm PLANOVER above.

Note in particular that our algorithms are easily implemented on top of existing DR-planners (provided their DR-plans satisfy the basic properties of Section 2.2.3 of the FA DR-plans). This follows from the fact that the entire information flow in our algorithms is expressed as attributes that can be stored and updated routinely as part of the DR-plan, such as: relevant cluster in the DR-plan for a given edge, relevant edges for a given cluster, the contact points that they define, and the inheritance of these contact points.

4.2 Illustrative Examples

Consider the 2d constraint system of Figure 3 and Algorithm OVERLIST. Figure 18 illustrates the key points of the algorithm for Graph $G1$.

In the case of the graph $G2$ of Figure 13(left) (formed from $G1$ by changing the edge h), the DR-plan $D(G2) = D(G1)$ and also the cluster $S(D(G2)) = S(D(G1))$. However, due to the change of h, UNRAVEL($C21$) directly calls UNRAVEL($C20$); unlike in the case of $G1$ UNRAVEL is not called for $C14$ and $C17$. The new $L(G2)$ turns out to be all edges in subgraphs corrsponding to $C4$ and $C20$.

In the case of the graph $G3$ Figure 13(right) (formed from $G1$ by changing the edge e), the DR-plan $D(G3) = D(G1)$ and also the cluster $S(D(G3)) = S(D(G1))$. However, due to the change of e, UNRAVEL($C6$) gets called unlike in the case of $G1$ and $G2$. The new $L(G3)$ turns out to be all edges in original graph $G3$.

In the case of the graph $G4$ Figure 14 (formed from $G1$ by changing the edges i and e), the DR-plan $D(G4)$ output by FA must be different from $D(G1)$ due to the properties in Section 2.2.3. However, the cluster $S(D(G4)) = S(D(G1))$. In this case, UNRAVEL($C7$) directly calls UNRAVEL($C4'$); UNRAVEL is not called for singleton clusters (vertices) 21 and 22. Thus edges $(17, 21), (18, 22), (21, 22), (22, 23)$ are excluded and the edge i changed in $L(G3)$ to give $L(G4)$.

In the case of the graph $G5$ Figure 15 (formed from $G1$ by differently changing the edges i and e), the DR-plan $D(G5)$ output by FA must be different from $D(G1)$ and $D(G4)$, due to the properties in Section 2.2.3. However, the cluster $S(D(G5)) = S(D(G1))$. In this case, UNRAVEL($C7$) calls both UNRAVEL($C4''$) and UNRAVEL($C5$). $L(G5)$ is again $L(G3)$, with i changed.

In the case of the graph $G6$ Figure 16 (formed from $G1$ by differently changing the edges g and e), the DR-plan $D(G6)$ output by FA must be different from $D(G1)$, due to the properties in Section 2.2.3 and moreover, $S(D(G6)) = C7'$ is not the same as $S(D(G1))$. In this case, UNRAVEL($C7'$) calls both UNRAVEL(12) and UNRAVEL($C7$), which in turn directly calls UNRAVEL($C4$). $P_{C7'}$ is the set $\{19, 20, 15, 12\}$. $L(G6)$ is exactly the set of edges in the subgraph of $G6$ corresponding to the cluster $C4$, along with the changed edges e, f, g.

These examples show 2d (non triangle decomposable) geometric constraint systems for the sake of exposition. The general decomposition capabilities of the FA DR-planner permits Algorithm OVERLIST to handle general 3d systems as well (try the software in [34]).

4.2.1 A Note on Implementation

The FA DR-plans for the examples given here and many other geometric constraint systems were obtained by running our FRONTIER geometric constraint solver. See for example Figure 4. These and other examples – both 2d and 3d, and involving other types of objects and constraints besides points and distances – are available at the FRONTIER public domain site, where the source code can also be downloaded. The algorithms described here will be incorporated into FRONTIER's update mode, see Figure 6(right).

4.3 Proof of Correctness and Complexity

The correctness of Algorithms OVERLIST and PLANOVER relies crucially on the properties of the FA DR-plan given in Section 2.2.3. This is elucidated by the control flow and the text in Figure 17, which distinguishes between the variants in Figures 13, 14, 15 and 16.

The following theorem shows that Requirements 1 and 2 of the problem statement in Section 2.4 are met by algorithm OVERLIST. Requirement 3 was discussed in Section 4.1.

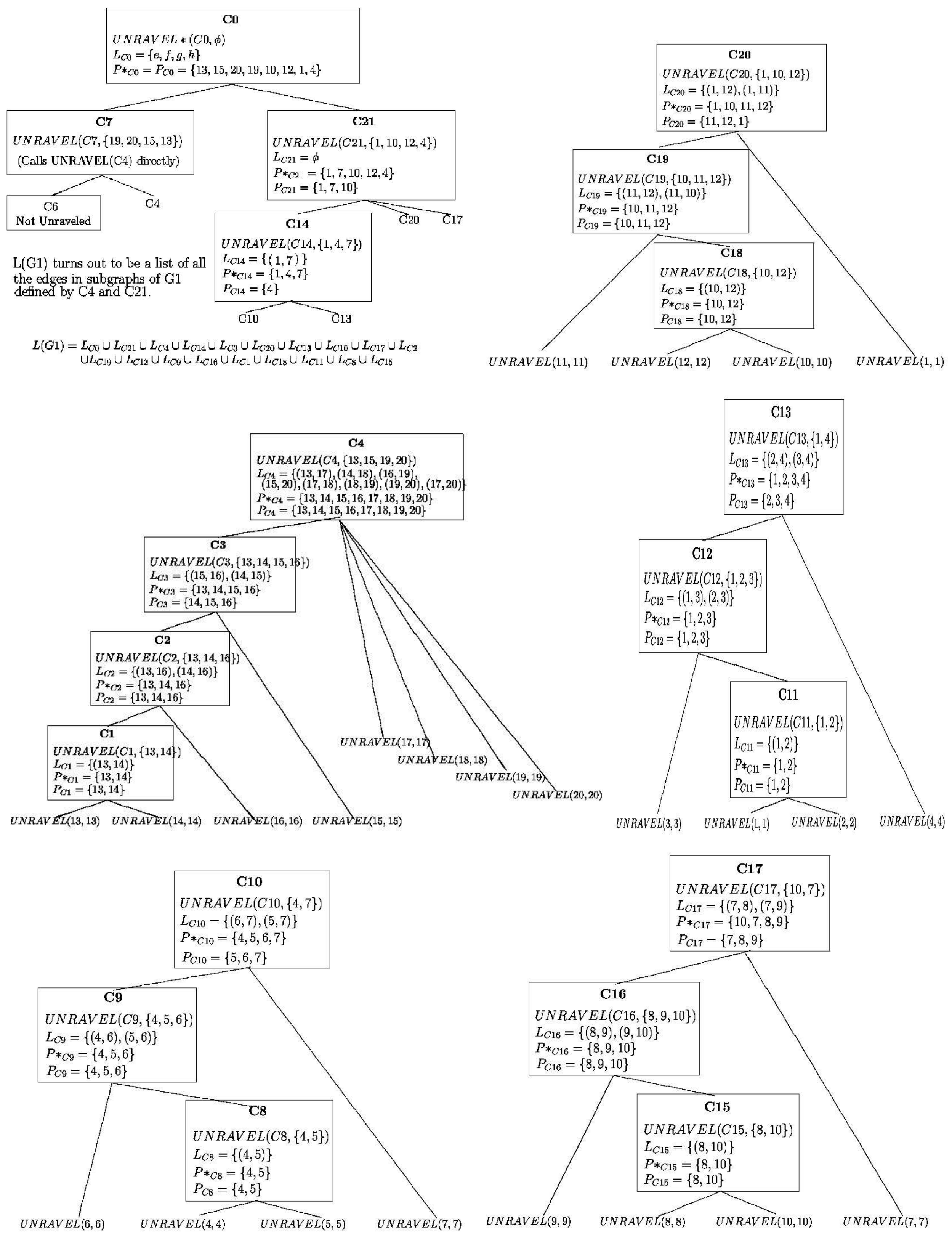


Figure 18: Working of OVERLIST showing calls to UNRAVEL for the graph $G1$ and DR-plan of Figure 3;

Theorem 4.2 *Let L_C be any sublist of edges output during a call to* UNRAVEL* *(no sublist L_C is output without such a call) and $L(G)$ be the union of all of these sublists as output by* PLANOVER, *when the algorithm* OVERLIST *is run on an input constraint graph G. Then the following hold.*

1. *Every edge in L_C represents a relevant explicit constraint for resolving C from its child clusters in the FA DR-plan $D(G)$ (output in Step 1 of* OVERLIST*). If a cluster C is an ancestor of a cluster D, then the edges in L_C (if any) are output before the edges in L_D (if any).*

2. *Every edge in L_C belongs to the (unique) minimal, nontrivial 1-(well)-overconstrained subgraph of G (and is hence reducible in G by Fact 2.2).*

3. *Any edge that is not in the complete output list $L(G)$ (i.e, it is not in any of the sublists L_C) is not reducible in G.*

Proof: Item 1 follows from the definition of the L_C in UNRAVEL* and from the recursive nature of UNRAVEL.

Proof of Item 2. We will show that every 1-overconstrained subgraph W of G contains every edge in L_C. This implies Item 2.
By the properties of FA in Section 2.2.3, the subgraph G_S of G corresponding to the cluster $S(D(G))$ is 1-well-overconstrained and contains the minimal 1-well-overconstrained graph.
Hence it is sufficient to show that every 1-(well)-overconstrained subgraph W of G_S contains every edge in L_C. This follows from Item (iv) in Claim 1 below.

Claim 1: *Let C be any cluster on which* UNRAVEL *is called, let G_C be the corresponding subgraph of G.*

(i) Any 1-well-overconstrained subgraph W of G_S must intersect G_C on a wellconstrained or well-overconstrained subgraph W_C

(ii) Let G_{C_i} be the subgraphs of G corresponding to the children C_i of C, found in $T_C^{-1}(C)$. The intersection $W_{C_i} = W_C \cap G_{C_i} = W \cap G_{C_i}$, if nonempty, is wellconstrained and nontrivial provided G_{C_i} is nontrivial. (the trivial cases of G_{C_i} are handled straightforwardly).

(iii) (a) Either $W_C \subseteq G_{C_i}$, i.e, $W_C = W_{C_i}$, for some i, or
(b) W_{C_i} is nonempty for every child cluster C_i of C.

(iv) If UNRAVEL* *is called at C, then (iii)(b) holds; i.e, W_{C_i} is nonempty for every child cluster C_i of C and, more importantly, W_C includes all edges in L_C and every contact point in P_C.*

Proof. We prove Claim 1 by induction on the number l_C of clusters along a path p in the DR-plan $D(G)$ from $S(D(G))$ to C on which UNRAVEL is called. This path consists of clusters that are both descendants of $S(D(G))$ and ancestors of C. It is clear from the algorithm that if UNRAVEL(*) is eventually called on C, then such a unique *unravelling path* p must exist, and UNRAVEL is called on all of the clusters on this path.

Basis: $l_C = 0$, i.e, $C = S(D(G))$. In this case, $W_C = W$, so (i) holds immediately, and in fact, W_C is well-overconstrained. To prove (ii), (iii), (iv), we first prove the following claims.

Claim 2:
(a) W_C cannot be a subgraph (proper or not) of any of G_{C_i}
(b) If some W_{C_i} is nonempty, then it cannot be overconstrained unless it is also trivial.

Proof. If W were a subgraph of one of the G_{C_i}'s or if W_{C_i} was overconstrained, then G_{C_i} would be well-overconstrained, and by definition of $S(D(G))$, we would contradict our Case 1 assumption that C_i's parent $C = S(D(G))$. This proves Claim 2.

Claim 3: *None of the W_{C_i}'s is trivial overconstrained unless the corresponding G_{C_i} is trivial overconstrained.*

Proof. Assume not. This implies that G_S contains a well-overconstrained subgraph W_C that intersects at least one of G_{C_i} on a trivial overconstrained subgraph. However, by a key property of FA DR-plans in Section 2.2.3, such a subgraph would have to directly yield or be represented as one of the child clusters of C, contradicting Claim 2a. This proves Claim 3.

Claim 2 and Claim 3 imply that there are at least 2 nonempty W_{C_i} and if nonempty, then the W_{C_i} are well or underconstrained and nontrivial provided G_{C_i} is nontrivial.

Next we show that the W_{C_i} are nonempty for every child cluster C_i. Suppose to the contrary that W_C overlaps only the G_{C_i} for $i \in Q$, where Q is a proper (index) subset of the children of C and $|Q| > 1$. Let L_C^Q be the restriction of L_C induced by Q, i.e, those edges in L_C that are incident on only vertices in the G_{C_i} for $i \in Q$. Let G_S^Q be the subgraph of G_S induced just by the vertices in $G_{C_i} : i \in Q$. Now: density(G_S^Q) is at least $\sum_{i \in Q}$ density(G_{C_i})+ (total edge weight of L_C^Q) which is at least $\sum_i$ density(W_{C_i})+ (total edge weight of L_C^Q) which is at least density(W_C). But since W_C was chosen to be overconstrained, this implies that G_S^Q is overconstrained. Since we know that none of the G_{C_i} is overconstrained (recall that $C = S(D(G))$), by the properties of the FA DR-planner in Section 2.2, it follows that the C_i's have the same (wellconstrained) density as the G_{C_i}'s and it would further follow that the subgraph of $T_C^{-1}(C)$ induced by the $C_i : i \in Q$ is also overconstrained). But this contradicts the minimality property in Section 2.2.3 of the cluster C as having been found by an FA DR-planner.

Next we show that none of the W_{C_i} is underconstrained and all the contact points in P_C are contained in W_C. Assume to the contrary that one of these does not hold. It is clear that one of the inequalities (*) and (**) below must be proper: density(W_C) is $\leq$ **(*)** $\sum_i$ density(W_{C_i})+ (total edge weight of L_C) which is $\leq$ **(**)** $\sum_i$ density(G_{C_i})+ (total edge weight of L_C) (since the G_{C_i} are well or well-overconstrained by definition of DR-plan and properties of FA in Section 2.2.3, and since the W_{C_i} are not overconstrained). This latter quantity is is equal to density(G_S). But since G_S is just 1-overconstrained, it follows that $W_C = W$ would not be overconstrained which contradicts our choice of W as an overconstrained subgraph of G_S.
This completes the proof of the induction basis.

Inductive step: Assume Claim 1 is true for clusters C with $l_C \leq m$. For a cluster C with $l_C = m + 1$, there must be some other closest ancestor cluster E_0 on the path p with $l_{E_0} \leq m$ on which UNRAVEL* is called. Let $E_0, E_1, E_2, \ldots, E_k = C$, $1 \leq k \leq (l_C - l_{E_0})$ be the clusters along the path p between E and C.

By the induction hypothesis on E, W_{E_1} is nonempty, wellconstrained, nontrivial (unless G_{E_1} is trivial) and includes all the contact points $P_{E_0}^* \cap G_{E_1}$. In fact, all of these contact points must lie in (be inherited by) $G_{E_1}, \ldots, G_C$: this is clearly true if $k = 1$ and $E_1 = C$. If not, i.e, if $k > 1$, then this is true again because otherwise UNRAVEL* would have been called on one of the clusters $E_1, \ldots, E_{k-1}$, contradicting the choice of $E = E_0$ as closest ancestor to C where UNRAVEL* is called.

This means that $W_{E_1} \cap G_C$ is nonempty and hence: This means that W_C defined as $W \cap G_C = W_{E_1} \cap G_C$ (since $G_C \subseteq G_{E_1}$), is also nonempty. Therefore by the induction hypothesis (ii) applied to its parent E_{k-1}, it follows that (i) holds for W_C.

To prove (ii), (iii), (iv) of the induction step, we consider two cases. Whether UNRAVEL* is called at C or not. In the former case, consider again the contact points $P_{E_0}^*$ inherited from C's closest ancestor E_0 on p where UNRAVEL* is called. As before in the inductive step for (i), W_C must contain all of the contact points $P_{E_0}^* \cap G_{E_1}$, where E_1 is a child of E_0. Since UNRAVEL* is called at C, W_C must therefore have a nonempty intersection with at least 2 of C's children C_i. This is a different proof of Claim 2a of the induction basis. The remainder of the inductive step for this case goes through exactly as in the induction basis.

In the latter case, i.e, when UNRAVEL* is not called at C, Claim 2a is now false, and so is Claim 3. However in this case we do not need to show (iv). To show (ii) and (iii), we modify Claim 2a and Claim 3 as follows. The proofs of these modifications are straightforward from the fact that UNRAVEL* is not called at C.

Claim 2a': *W_C is a subgraph of G_{C_i} for at least one of its children C_i, only if all the contact points P_E^* of C's parent E fall into C_i.*

Claim 3': *If W_C is not a subgraph of some G_{C_i}, then none of the W_{C_i}'s is trivial overconstrained unless the corresponding G_{C_i} is trivial overconstrained.*

The remainder of the proof for this case proceeds exactly as in the induction basis. □(Item 2)

Proof of Item 3 There are only 2 cases of these edges that are left out of $L(G)$: those that are in G_S, i.e, the subgraph corresponding to $S(D(G))$ and those that are not. Edges outside G_S are not reducible by Section 2.2.3, so that takes care of the latter case.

The former case of edges inside G_S must appear as unchanged edges in $T_C^{-1}(C)$, i.e, relevant edges for resolving a unique descendant cluster C of $S(D(G))$ for which UNRAVEL* is not called. These are the clusters

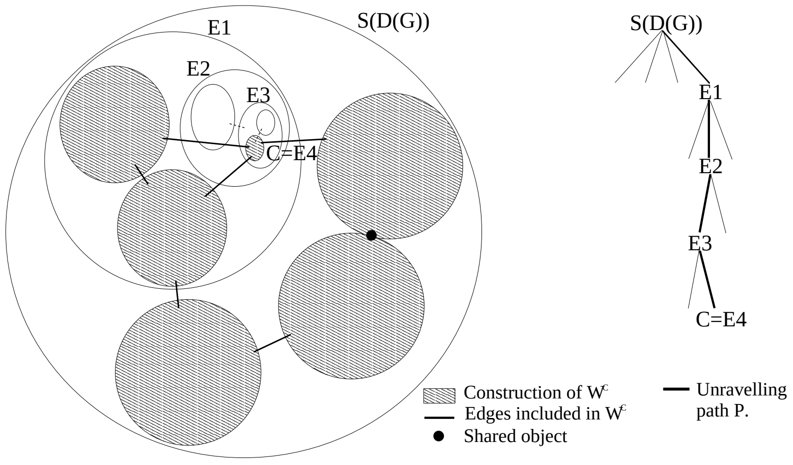


Figure 19: Construction of W^C for cluster C

C in which *all* contact points P^*_E - inherited through UNRAVEL on its parent E - fall into G_{C_i} for one of the child clusters C_i of C.

For each such cluster C, we exhibit a subgraph W^C of G_S that is 1-overconstrained and does not include any of the unchanged edges of G that are incident on more than 1 cluster in $T^{-1}_C(C)$. This implies that none of these edges belong to the minimal 1-overconstrained subgraph of G and by Fact 2.2 are not reducible.

This W^C is constructed as follows. See Figure 19. Take the unique unravelling path p (defined during the proof of Item 2) between $S(D(G)) = E_0, E_1, \ldots E_k = C$ $(k \geq 1)$ in the DR-plan $D(G)$. Walking down p increment W^C in stages, one stage for every cluster E_l on p where UNRAVEL* is called. Include into W^C all the subgraphs of all the children clusters of E_l except for the ancestor of C, namely E_{l+1}. Include into W^C all the unchanged edges of G that are incident on more than 1 cluster in $T^{-1}_{E_i}(E_i)$ as well as the contact points in $P^*_{E_l}$ (unchanged vertices of G) that are in E_{l+1} – we call this the set L of *dangling* contact points. Finally, include all the edges in G_{C_i} into W^C. By construction, W^C contains none of the unchanged edges that are incident on more than 1 cluster in $T^{-1}_C(C)$.

The argument for W^C being 1-overconstrained runs as follows. By the fact that G_S is 1-overconstrained, it follows that at any given stage l of incrementation, W^C can be made 1-overconstrained by embedding the set L of dangling contact points (i.e, adding enough edge weight to make them) into a well-constrained subgraph. At the final stage, observe that we perform exactly this embedding by including all the edges in the well-constrained G_{C_i} into W^C. □(Item 3, and Theorem)

4.3.1 Complexity

The complexity of constructing the FA DR-plan $D(G)$ for G is given in Section 2.2.3 and by Figure 17, it gets included into the complexity of Algorithm OVERLIST: it represents the dominant complexity term.

However, the key point was to incrementally and flexibly output reducible constraints, given an *already existing* DR-plan $D(G)$. I.e, the crucial facility of Algorithm PLANOVER (besides its generality) is the ability to efficiently output sublists of reducible constraints in "latest-solved-first" order and to output on demand the relevant reducible constraint sublist for a particular cluster (see Section 4.1). It is the complexity of these procedures that are of interest. In other words, *given a cluster C, what is the complexity of* UNRAVEL *for outputting the list L_C, if one exists, as a function of the standard graph and DR-plan parameters, and the depth of C in the DR-tree.*

The answer depends on whether and how standard information such as: relevant edges of a cluster, contact points of a cluster and so on are maintained as part of the DR-plan. To isolate this aspect, let us assume the complexity of processing *at* any given cluster C, i.e, computing L_C, P_C, $P^*_C \cap C_i$ etc. (not including the recursive call) during UNRAVEL(C) is $t(k_C, r_C)$. Here, the quantity k_C is the number of child clusters in $T^{-1}_C(C)$ that participate in C, the fan-in of the DR-plan at C, and is bounded by $|V|$; r_C is the number of relevant edges in C.

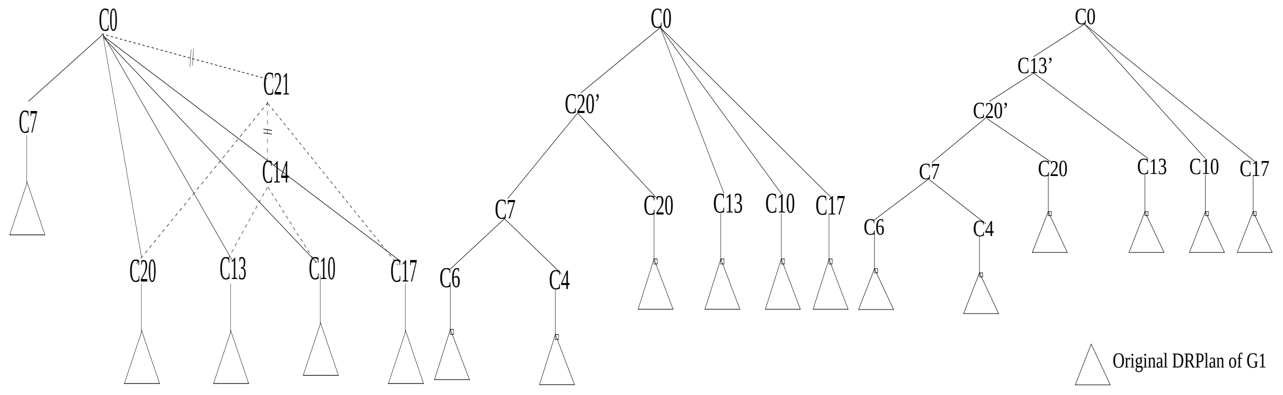


Figure 20: Left: unoptimized DR-plan of $G1$ in Figure 3, after reduction of edge $(1, 7)$; unravelling path is marked; Middle and Right: two optimization steps

Note: Our complexity analysis assumes that t is a linear function – simple graph data structures ensure this, even if contact points and relevant edges have to be computed from scratch *without* being maintained as part of the DR-plan. If these data are maintained systematically using high-efficiency data structures, then t could be a *polylogarithmic* function, decreasing the following bound significantly. We also assume that each edge in the hypergraph G has a bounded arity (2 in a usual graph), otherwise, the arity enters the complexity in a straightforward manner as a linear factor.

Let l_C represent the length of (number of clusters B on) the unravelling path p_C (in $D(G)$) from $S(D(G))$ to C, which is bounded by $|V|$ (due to properties in Section 2.2.3). Hence the complexity of outputting the list L_C (if one exists) – given a graph G, DR-plan information $D(G)$ and $S(D(G))$ and a cluster C – is bounded by: $\sum_{B \in p_C} t(k_B, r_B)$. This clearly grows with l_C, according to Requirement 2 of Section 2.4. One rough complexity upper bound exhibiting this dependence is $O(l_C(|V|+|E|))$. However, this is a loose upper bound since no edge of the graph and no cluster in the DR-plan is ever inspected more than once, i.e, since for any C, $\sum_{B \in p_C} r_B$ does not exceed $|E|$, and since $\sum_{B \in p_C} k_B$ does not exceed $|V|^2$ (the number of clusters generated by FA DR-planner does not exceed $O(|V|^2)$, by Section 2.2.3). Thus the complexity of interest does not exceed $O(\max\{|V|^2, |E|\})$. Combining the two bounds, the complexity of interest is bounded by: $O(\min\{l_C(|V| + |E|), \max\{|V|^2, |E|\}\})$, even without prior DR-plan maintanence of contact point and relevant cluster information. In practice (see examples in Section 4.2) – this bound does not exceed $O(|E|)$.

4.4 Updating the DR-plan After Edge Reduction

We now reconsider Requirement 4 of Section 2.4. If the weight of one of the reducible edges in L_C for a cluster C is reduced by one, the DR plan $D(G)$ may no longer be correct. How can the DR-plan $D(G)$ be modified efficiently to give an correct DR-plan? The modified plan should be near-optimal, but preferably be found without reorganizing any of the (maximal) descendant clusters F of $S(D(G))$ whose subgraphs G_F remained wellconstrained after the reduction. This can be done using the relevant constraint lists L_F and contact points along with inheritance information P_F and P_F^*, that are used during UNRAVEL. As discussed in Section 4.3.1, they can be maintained easily as standard information along with the clusters in the DR-plan. We leave out the formal description and provide an intuitive description using pictures in Figure 20.

When one of the edges in L_C is weight-reduced by 1, the clusters that are no longer wellconstrained are exactly the clusters E along the unravelling path p in $D(G)$ from $S(D(G))$ to C, excluding $S(D(G))$, whose corresponding subgraph G_S was previously 1-well-overconstrained and now becomes wellconstrained. All other clusters are preserved. The maximal preserved sub-DR-plans are rooted in the clusters F which are exactly the children of the clusters E, excepting the children that are directly on p, marked on the left of Figure 20. The old DR-plan and the children on p are shown as dashed.

An initial valid but unoptimized DR-plan attaches all of the clusters F directly as the children of $S(D(G))$, however if there are many such clusters, such a plan could cause $S(D(G))$ to have a large fan in, i.e, the system to be solved in order to resolve or recombine $S(D(G))$ could be large, making the new DR-plan far from optimal.

Now the contact points inherited by the clusters F can be used to optimize this initial DR-plan. The contact points are read off from the P_E^* for the ancestors E, of the clusters F, on the unravelling path p, including the cluster $S(D(G))$. The contact points should be considered in an oldest-first order; i.e, the contact points inherited from $S(D(G))$ are first to be considered in recombining the clusters F. This is because intuitively, $S(D(G))$ was the only cluster whose subgraph G_S was previously overconstrained, and hence any "compensation" for the reduction of the edge from L_C should start with the relevant edges and contact points at $S(D(G))$, and propagate downwards along p. Successive steps in obtaining an optimized DR-plan are described in the sequence of 2 pictures to the middle and right of Figure 20.

4.5 Extending to k-overconstrained graphs

One important observation is that Fact 2.2 extends to give a definition of a minimal, unique k-overconstrained subgraph of a given k-overconstrained graph. Thus the 1-overconstraint method given here directly yields *one* reasonable way of dealing incrementally with a k-overconstrained graph as follows: supply the user with k sets of reducible constraints, in succesive stages, such that each set is, in a well-defined sense complete and minimal for its stage: at the ith stage, the reduction of a constraint from the output set will ensure that the graph is atmost $k - i$-overconstrained and has not become underconstrained.

5 Summary and an Open Problem

A geometric constraint solver algorithm employs highly specialized mathematics and occupies a central position in many applications in computer-aided mechanical design. In most cases, the constraint solver is embedded in other software and is not directly exposed to the user.

When formulating a constraint problem in an application context, it is possible that situations arise that require the user to interact with the solver. The required interaction may be a problem reformulation or assistance to the solver to select a different solution variant. In those situations, intuitive methodologies must be developed for users who are unfamiliar with the underlying mathematics and with the particular strategies employed by the constraint solver algorithms. Such users must be able to effectively communicate their requirements without having to understand the workings of the solver or the deeper principles on which it is based.

We have considered the issues requiring user interaction for overconstrained problems, where users may not understand why their problem specification has redundant constraints. Here, we must present the user with all relevant choices for deleting redundancies. Currently, solvers either give no information, simply labelling the problem as overconstrained, or else flag only an effectively random subset of the reducible constraints at the immediate cluster in which the problem was detected. But the constraints that have been identified may be essential to application requirements. Thus, *all* relevant constraints must be identified.

Moreover, the constraints that have been identified must be output efficiently and flexibly as demanded by the user, from particular subsystems of constraints that represent specific parts or features where the user has design flexibility. Furthermore, the algorithm should efficiently use information that is central to the constraint solver, and hence already exists, such as the DR-plan. Such data must also be updated efficiently or maintained dynamically when one of the reducible constraints is removed by the user. We have formally identified these requirements, and given efficient algorithms both for simple, but practical situations and for the general case as well.

In this paper, we dealt with the (general 2d and 3d) cases of 1-overconstrained problems. Based on it, we suggested an incremental method of dealing with k-overconstrained problem. However, the problem with the general k-overconstrained case is that a formal definition of it appears to be significantly more complex (there are many different, reasonable definitions), based on preliminary work in [35]. When placing constraints incrementally during the design, the 1-overconstrained algorithm is sufficient. When combining already constrained designs, or when editing them, the k-overconstrained case can arise. The incremental method, offered in Section 4.5, for dealing with the k-overconstrained case may not be adequate in some situations, where the the user might potentially want to view and choose from entire k-tuples of constraints to reduce or delete, rather than being offered reducible sets to choose from incrementally, in a prescribed order.

Acknowledgements

Many thanks to Nandhini Kohareswaran for the pictures.

References

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

[2] W. Bouma, I. Fudos, C. M. Hoffmann, J. Cai, and R. Paige. A geometric constraint solver. *CAD*, 27:487–501, 1995.

[3] B. Bruderlin. Constructing three-dimensional geometric object defined by constraints. Implementing FRONTIER: a geometric constraint solver Master's thesis; Technical report, University of Florida, CISE departm ent, 2001.

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

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

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

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

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

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

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

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

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

[13] C. M. Hoffmann, A. Lomonosov, and M. Sitharam. Decomposition plans for geometric constraint problems, Part I: performance measures for CAD. *JSC*, 31:367–408, 2001.

[14] C. M. Hoffmann, A. Lomonosov, and M. Sitharam. Decomposition plans for geometric constraint problems, Part II: new algorithms. *JSC*, 31:409–428, 2001.

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

[16] C. M. Hoffmann and J. Peters. Geometric constraints for CAGD. In M. Daehlen, T. Lyche, and Schumaker L., editors, *Mathematical Methods for Curves and Surfaces*, pages 237–254, 1995.

[17] C. M. Hoffmann and R. Vermeer. Geometric constraint solving in R^2 and R^3. In Du D. Z. and Hwang F., editors, *Computing in Euclidean Geometry*, pages 266–298, 1995.

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

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

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

[21] R. S. Latham and A. E. Middleditch. Connectivity analysis: a tool for processing geometric constraints. *CAD*, 28:917–928, 1996.

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

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

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

[25] J. J. Oung and M. Sitharam. A fast, simple solver for constraints without parameters. Technical report, University of Florida, CISE department, 2001.

[26] J. J. Oung. Implementation of a geometric constraint solver: FRONTIER. Master's thesis, Technical report, University of Florida, CISE department, 2001.

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

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

[29] D-Cubed. *Commercial Constraint Solver.* In *http://www.d-cubed.com.*

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

[31] M. Sitharam, J. Oung, A. Arbree, N. Kohareswaran. FRONTIER: enabling constraints for feature-based modeling and assembly. *Abstract in ACM-Solid Modeling* 2001.

[32] M. Sitharam, J. Oung, A. Arbree, N. Kohareswaran. FRONTIER, a 3d Geometric Constraint Solver Part I: Contributions and Design. *Submitted.* condensed version in *http://www.cise.ufl.edu/∼sitharam/full-frontier.pdf.*

[33] M. Sitharam, J. Oung, A. Arbree, N. Kohareswaran. FRONTIER, a 3d Geometric Constraint Solver Part II: Algorithms. *Submitted.* condensed version in *http://www.cise.ufl.edu/∼sitharam/full-frontier.pdf*

[34] M. Sitharam. GNU Public License: FRONTIER, a 3d Geometric Constraint Solver. *http://www.cise.ufl.edu/∼sitharam*

[35] M. Sitharam, C.M. Hoffmann. The general geometric overconstraint problem. *In preparation*

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

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

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

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

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