Making Constraint Solvers More Usable: Overconstraint Problem

Christoph M. Hoffmann* Meera Sitharam† Bo Yuan*

June 12, 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 defined 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 a 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 represenation. 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 [35, 20, 34, 26, 18, 32, 1, 3, 15, 16, 9, 10, 11, 12, 13, 17, 8, 14, 6, 7, 23, 24, 2, 25]. 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 [10, 11, 12, 13, 2, 25], including generalize at varying degrees 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.

*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*

The solution of a geometric constraint system is a real zero of the corresponding algebraic system. In other words, the solution is a class of valid instantiations of the geometric elements such that all constraints are satisfied. Here, it is understood that such a solution is in a particular geometry, for example the Euclidean plane, the sphere, or Euclidean 3 dimensional space. For recent reviews of the extensive literature on geometric constraint solving see, e.g, [12, 19].

Geometric constraint solvers are constructed to meet 3 competing challenges.

1. Generality of expression;
2. Efficiency of realization; and
3. Resolution of ambiguities or inconsistencies, and updating (dynamic maintanence).

For instance, the latter problem stands in a tradeoff relationship with the first. It worsens with the increase in expressive power or generality of constraint systems which may cause them to have multiple solutions or realizations, conflicting requirements, redundancies, etc. while the constraint solver is not able to reconcile conflicts and eliminate in a clever way. Thus, 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 over-constrained problems have to somehow be detected and turned into wellconstrained problems, with intervention by the designer.

When such intervention is required, the constraint solver should offer guidance by presenting viable choices. These choicenitemize

- should not require mathematical proficiency on the user's part;
- they should neither be limited arbitrarily, nor should they include choices that are irrelevant to the core problem;
- and they should be unique in a well-defined sense so that the user can expect repeatability of the set of choices.

Efficient Realization needs DR Plans

A good decomposition of the geometric constraint system is indispensable in dealing with all three of the challenges listed above. Consider the first: i.e, guaranteeing efficiency of realizing the constraint system, while maintaining full generality of expression. The cost of solving a geometric constraint system is directly proportional to the size of the largest subsystem that is solved using a direct algebraic/numeric solver. This size dictates the practical utility of the overall constraint solver, since the time complexity of the constraint solver is at least exponential in the size of the largest such subsystem. Hence the *optimal* or most efficient decomposition would minimize the size of the largest such subsystem. In other words, any geometric constraint solver should first solve the problem of efficiently finding a close-to-optimal *decomposition-recombination (DR) plan*, because that dictates the viability of the constraint solver. Finding a DR-plan can be done as a pre-processing step by the constraint solver: a robust DR-plan would remain unchanged even as minor changes to numerical parameters or other such on-line perturbations to the constraint system are made during the design process.

While DR-plans were informally used by many constraint solvers, the formal definition of a DR-plan (given in Section 2.1) as well as the various performance measures - based on the above two challenges as well as others - were first given in [12]. A new DR-planner called the Frontier vertex algorithm (FA) was designed in [13] to optimize these performance measures. The FA DR-planner underlies FRONTIER [27, 28, 29, 22] (available as GNU software) [30], which is to our knowledge the only constraint solver that systematically deals with fully general, 3d constraint systems.

Informally, a geometric constraint solver which solves a large constraint system E uses a a DR-planner to guide a direct algebraic/numeric solver - which is restricted to solving small subsystems - as follows. It proceeds by repeatedly applying the following three steps at each iteration i.

1. Find a small solvable subsystem S_i of the (current) entire system E_i (at the first iteration, this is simply the given constraint system E, i.e, $E_1 = E$). This step is indicated by a rectangle in Figure 1. Subsystem S_i could be also chosen by the designer.

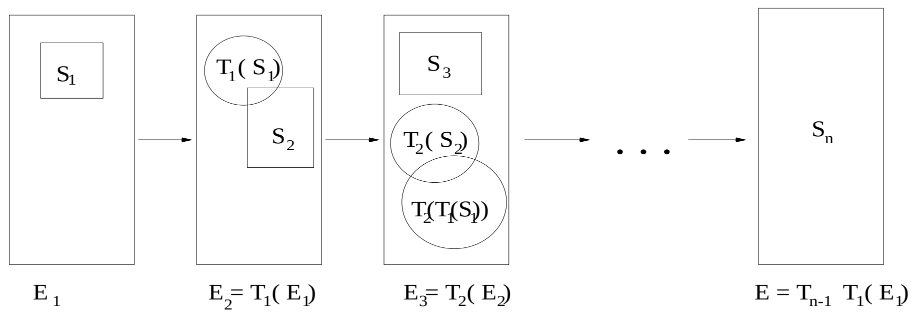


Figure 1: Solving a well-constrained system by decomposition and recombination

2. Solve S_i using the direct algebraic/numeric solver.
3. Using the output of the solver, and perhaps using the designer's help to eliminate some solutions, replace S_i by an *abstraction or simplification* $T_i(S_i)$ thereby replacing the entire system E_i by a simplification $T_i(E_i) = E_{i+1}$. This step is indicated by an oval in Figure 1. Some informal requirements on the simplifiers T_i are the following: we would like E_i to be (real algebraically) inferable from E_{i+1}; i.e, we would like any real solution of E_{i+1} to be a solution of E_i as well.

This solver terminates when the small, solvable subsystem S_i found in Step 1 is (a representation of) the entire algebraic system E_i. An optimal DR-plan will minimize the size of the largest S_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 children represent the different subsystems (found earlier) that were combined to form S_i. If the whole system is underconstrained, the solver should still isolate and solve its maximal solvable subsystems.

Inconsistency, Ambiguity and Dynamic Maintanence using DR-plans

A good DR-plan is not only crucial for efficiently realizing constraint systems, in addition, a DR-plan helps the designer conceptually since it can be viewed as a feature hierarchy that decomposes the product being designed. In fact, DR-planners such as FA mentioned earlier [13], will take a conceptual feature hierarchy (input by the designer) and ensure that the output DR-plan incorporates it (is a proper refinement of it).

Therefore, since a DR-plan is generally available, it can and should be efficiently put to use for the second challenge listed above, i.e, dealing with ambiguities and inconsistencies through user intervention. The following specific problems arise in this context:
"how to deal with multiple solutions;" "how to isolate the generically under and overconstrained parts; and how to offer the user an incremental list of constraints to add and remove; the user should be permitted to specify which portions of the system these constraints should come from, and they should cause the least amount of additional work in updating the solution and the DR-plan"

These problems have been approached in the literature with various degrees of success. Many of these methods make effective use of an existing DR-plan [3, 4, 25, 1, 27, 30, 28, 29, 22], [1, 25], [27, 28, 29], [30].

Systematic methods also exist [30, 22, 28, 29] for dynamically maintaining or updating the DR-plan minimally (and hence for reducing the amount of repeat solving), when a change is made to the constraint system. Some kinds of updates are well understood for general, 3d constraint systems [30, 22, 28, 29], for example when the change involves a parameter value, added constraint, added object, changed feature hierarchy etc. However, it has so far not been well understood for the case of constraint removal, or deletion, which arises when overconstrained problems are to be corrected.

Main Contribution

Current solvers encourage users to delete redundant or contradictory constraints, since there are no attractive heuristics to do so automatically. Current solvers flag constraints found to be redundant or contradictory when they are first detected within a cluster being solved. However, which these constraints are, i.e, the cluster at which the overconstraint is first detected, depends on the particular DR-plan, and some of the best DR-planners are effectively nondeterministic. Hence such adhoc identification of constraints to delete is essentially random, non-repeatable and is unacceptable in practice.

In this paper, we investigate generically overconstrained problems. We first define precisely what is meant by a generically overconstrained problem, and how to define conceptually the *entire minimal subset* of constraints that are in conflict. We show that this subset is unique and well-defined, and offer a simple solution to the problem of isolating this subset, which works for fully general geometric constraint systems in arbitrary dimensions. We then point out the drawbacks of this solution: its relative inefficiency, since it ignores the DR-plan whose availablility can be assumed, and its inflexibility. We then give an algorithmic solution that is efficient - generally linear time in the number of *relevant, output* constraints; retains full generality and moreover:

- traverses the given DR-plan top down to incrementally output this unique set of constraints in reverse solving order – this would also minimize the need to resolve of already solved parts of the DR-plan;
- selects constraints from desired parts of the constraint system (representing product features) demanded by the user;
- isolates information that can be 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 output constraints is chosen by the user to be removed.

Implementation

The algorithms developed here will be implemented and incorporated into two fairly established geometric constraint solvers that have different emphases: the SketchWorks and Erep solvers developed at Purdue, the FRONTIER constraint solver [27, 28, 29, 22, 21], developed at Florida (the latter is available as GNU software [30]).

Organization

The paper falls naturally into two well-defined parts.

The first part (through Section 2) is devoted to the necessary background discussion for converging to an adequate formalization of the the problem. In particular, in Section 2, we develop a significant amount of background on geometric constraint graphs and DR-planners as well as crucial specifics of the Frontier Vertex DR-planner which relies on a network flow based, generalized degree of freedom analysis. We then give an an initial formalization of the problem, show that it is wellposed, give an initial, network flow-based algorithmic solution, discuss its drawbacks, and reformulate the problem according to the informal requirements listed in the introduction.

The second part of the paper gives an an efficient algorithmic solution to the final problem formulated in Section 2. Section 3 gives an algorithm for a commonly occuring class of 2d systems. Section 4 gives an algorithm and its correctness and complexity analysis, for fully general constraint systems in arbitrary dimensions. This general algorithm hides an intricate proof of correctness, which has been relegated to the Appendix. Conclusions are drawn in Section 5.

2 Background, Problem Statement, and Initial Solution

Given a constraint system, one usually constructs the DR-plan first, i.e, execute Steps 1 and 3 described in the Introduction, without access to an algebraic solver, i.e, one generates the entire DR-plan first before solving the subsystems.

To generate a DR-plan apriori, one would have to locate a solvable subsystem S_i, and without actually solving it, find a suitable abstraction or simplification of it that is substituted into the larger system E_i to obtain an overall simpler system E_{i+1} in Step 3. On the other hand, such a DR-plan would possess the advantage of being robust, or generically independent of particular numerical values attached to the constraints, and of the

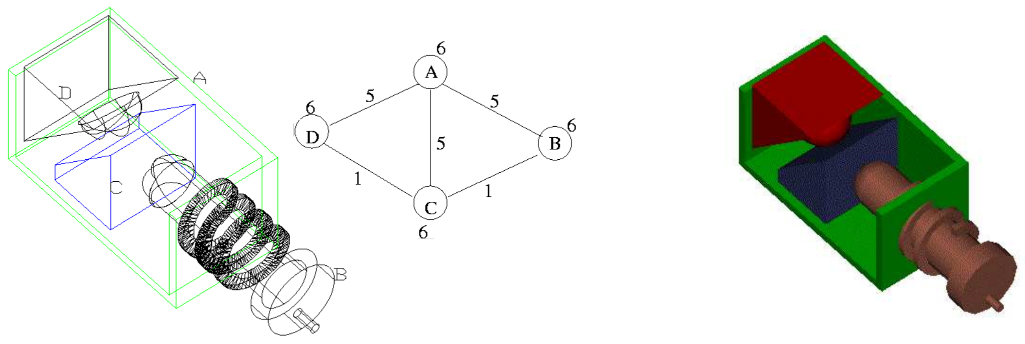


Figure 2: 3d assembly example with 1 extra degree of freedom, and corresponding constraint graph

solution to the S_i, and usually only depends on the degrees of freedom of the relevant geometric objects and geometric constraints.

In order to formally define such a DR-plan, we follow a common practice and view the constraint system as the constraint hypergraph: this abstraction permits us to build the DR-planner on the foundation of generalized degree of freedom analysis which is known to work well in estimating generic solvability of constraint systems *without* actually solving them. (This is explained more precisely after the formal graph-theoretic definitions are in place).

2.1 Constraint Graphs and Solvability

Recall that geometric constraint problem consists of a set of geometric elements and a set of constraints between them. A geometric constraint graph $G = (V, E, w)$ corresponding to geometric 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 integer 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, Figure 2 shows a 3d example with its constraint graph.

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

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

is called *dense*, where D is a dimension-dependent constant, to be described below. Function $d(A) = \sum_{e \in A} w(e) - \sum_{v \in A} w(v)$ is called *density* of a graph A.

The constant D is typically $\binom{d+1}{2}$ where d is the dimension. The constant D (whose negation is the density) captures the *dof* or degrees of freedom associated with the dense graph. For planar contexts and Euclidean geometry, we expect $D = 3$ and for spatial contexts $D = 6$, in general. If we expect the cluster 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.* It should be noted that certain *trivial* overconstrained graphs are common and require special treatment. These are: a single point in 2d or 3d (graphs with a singleton vertex of weight 2 or 3); and a pair of points and a distance constraint between them in 3d (graphs with 2 vertices of weight 3 and an edge of weight 1 between them). The special treatment is required because while these graphs are technically overconstrained (they have a smaller number of degrees of freedom than a rigid object), infact, this is because of their rotational symmetry. In other words, their "overconstraint" is an *external* overconstraint, affecting the objects that interact with them, rather than what is usually understood to be an overconstraint, i.e, between the objects internal to the subgraph.

Here we assume that the only rotationally symmetric systems we encounter correspond to the 2 trivial subgraphs listed above. See [28, 29] for the implications of rotationally symmetric systems and their corresponding subgraphs, and how they should be treated.

A graph that is dense and all of whose subgraphs (including itself) have density at most $-D$ is called *well-constrained.* 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 subgraphs, G remains dense. A dense graph is *minimal* if it has no dense proper subgraph. Note that all minimal dense subgraphs are well-constrained or well-overconstrained, but the converse is not the case. A graph that is not well-constrained or well-overconstrained is said to be *underconstrained.* Another equivalent definition of an underconstrained graph is one that contains a minimal cutset of (hyper)edges (a minimal set of edges whose removal disconnects the graph), of total weight less than D. However in the case where 1 (or k, $k > 2$ relevant only for hypergraphs) of the disconnected subgraphs resulting from the minimal cut have one rotational symmetry (or an external overconstraint, e.g, if it is a trivial dense graph), then the minimal cut needs to be of weight less than $D - 1(k)$ to cause the graph to be underconstrained.

Fact 2.1 *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 $-D$. In fact, a dense, nontrivial, underconstrained graph must contain an overconstrained, nontrivial proper subgraph. Viceversa, if a graph G of density $-D$ contained a nontrivial subgraph of density $-D+1$, then G must be underconstrained.*

In general, minimal dense subgraphs are allowed to include trivial dense graphs as proper subgraphs. Sometimes, a single dense edge of any kind is considered trivial and also treated as a special case, but this is more for reasons of efficiency. Generally, these do not require such special treatment.

While a generically solvable system always gives a well-constrained graph, the converse is not always the case. In fact, there are even minimal dense graphs whose corresponding systems are not generically solvable, and are in fact generically unsolvable (note that the position of the 'not' changes the meaning, the latter being stronger than the former). For a detailed discussion of genericity and the limits of purely combinatorial degree of freedom analysis, see [13].

However, due to the reasons discussed in the above paragraph, we restrict ourselves to a class of constraint systems where well or well-overconstrainedness of the constraint graph in fact implies the generic solvability of the constraint system. As pointed out, the converse is always true, with no assumptions on the constraint system.

Stated in terms of constraint graphs, the DR-planning problem involves finding a sequence of graphs G_i - a DR-plan traversed in a consistent linear order. The original constraint graph $G = G_1$ and every G_i contains a minimal well or well-overconstrained subgraph S_i, which is simplified or abstracted into a simpler subgraph $T_i(S_i)$ and substituted into G_i to give an overall simpler graph $G_{i+1} = T_i(G_i)$. While $T_i(S_i)$ should be simpler than S_i, it should also preserve essential information from S_i that is related to its interaction with the rest of G_i. If the original graph G_1 is wellconstrained, then the process terminates when $G_m = S_m$. (If not, the process terminates with the decomposition of G_m into a maximal set of well or well-overconstrained subgraphs).

NOTE: The linear ordering of simplification steps $G_i, G_{i+1}, \ldots$ etc. is not necessary in the definition of DR-plan, although they are often convenient for defining DR-planners. The DR-plan of a constraint graph G is inherently just a partial order, and can be represented as a directed acyclic graph, (which is typically for most part a tree or forest). DR-plans are *not* unique: see Figure 3.

An optimal DR-plan will minimize the size of the largest subgraph S_i found during the simplification process. I.e, it will *minimize the maximum fan-in* of the vertices in the DR-plan. By fan-in of a vertex we mean the number of children of the vertex. With this description, it should be clear, that DR-plans obtained using the weighted, constraint graph model are generically robust with respect to the changes made to the geometric constraints, as long as the number of degrees of freedom attached to the objects and destroyed by the constraints remains the same, the same DR-plan will work for the changed constraint system as well.

2.2 The Frontier Vertex Algorithm (FA-DR planner)

Prior to [12, 13], the DR-planning problem and the relevant performance measures for good DR-planners were not formally defined, and most DR planners were based on decomposing the graph (if possible) into fixed rigid

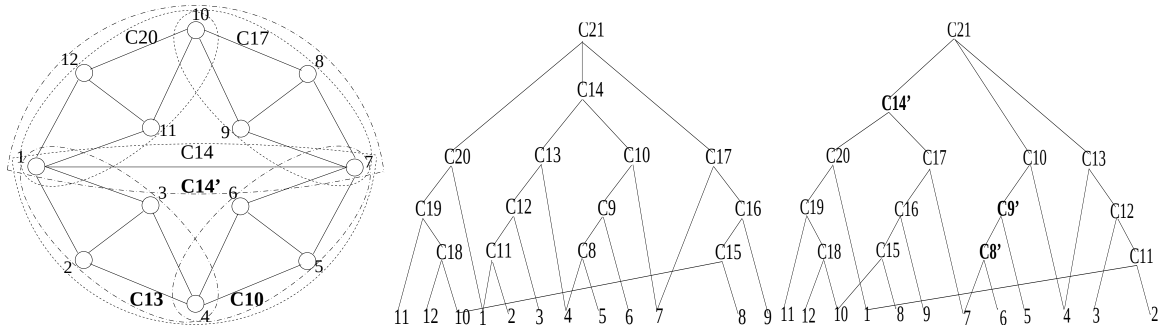


Figure 3: Geometric constraint graph showing wellconstrained subgraphs and 2 DR-plans: all vertices have weight 2 and edges weight 1

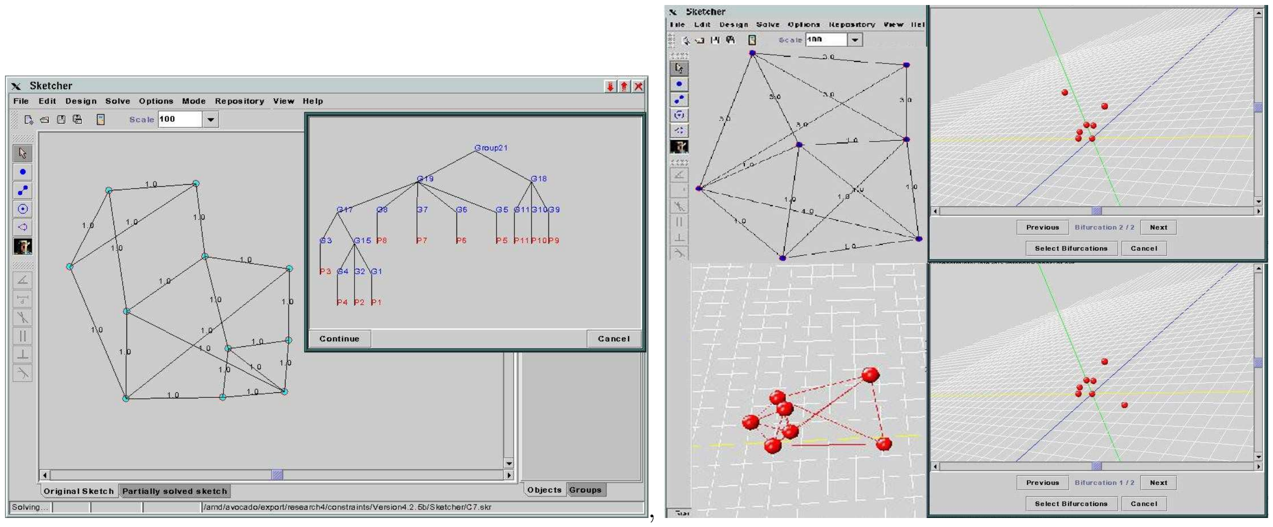


Figure 4: FRONTIER screenshots. Left: non triangle-decomposable constraint graph and DR-plan: all vertices have weight 2 and edges weight 1. Right: input sketch representing 3d constraint system, two candidate solutions and chosen one

patterns (i.e, recognizing shapes corresponding to certain specific minimal dense subgraphs, such as triangles of 3 points and 3 distance constraints). This works well in many cases, especially in 2d [5],[23, 24, 25]. This class of 2d constraint systems is commonly occuring and is generally refered to as *triangle decomposable*. However, many natural constraint systems are more complex (especially in 3d) and do not lend themselves to this approach. See, e.g, Figures 4 and 5.

In such cases, the graph needs to be decomposed into minimal dense subgraphs of arbitrary topology, their only defining property being their density and minimality. Prior to [13] approaches to such generalized degree of freedom analysis had several drawbacks with respect to the performance measures of good DR-planners defined in [12]. The DR-planner that was designed specifically in [13] to optimize these measures was the Frontier vertex Algorithm (FA) which works for general constraint hypergraphs representing geometric constraint systems in arbitrary dimensions. Note that while our description here is general, our illustrations, however, do not involve hypergraphs, and generally involve only 2d points and distance constraints.

FA is a recursive algorithm that is used to build the DR-plan. Each node or cluster C of the DR-plan is built using 2 steps that are perfomed alternately and repeatedly: (1) finding or isolating the minimal dense subgraphs (the decomposition step) and (2) simplifying or transforming it into the cluster C (the recombination step). Both steps are applied to a graph which consists of clusters already simplified in the previous iteration.

For Step 1, small, well-constrained subgraphs are isolated by finding minimal dense subgraphs. The algorithm in [10] which FRONTIER [27, 30] employs a simple but crucial modification of Network Flow, in particular, the

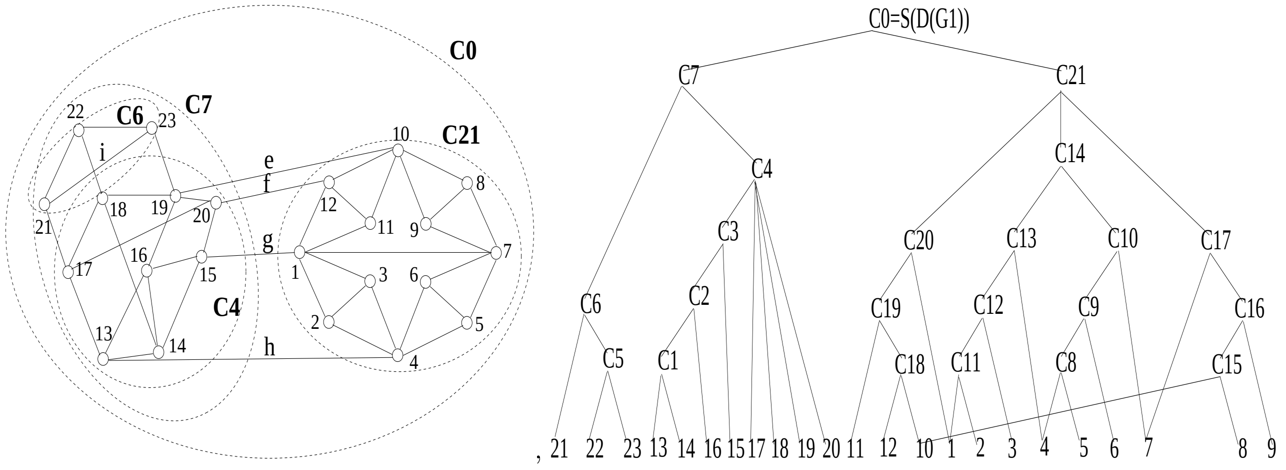


Figure 5: Main example constraint graph G1 and DR-plan all vertices have weight 2 and edges weight 1

incremental max flow algorithm to first isolate a dense subgraph G' (with chosen lower bound on density) of the given constraint graph G. Then a *minimal dense* subgraph inside it can be found by successively dropping vertices v and redoing the flow inside G'. If a new dense subgraph G'' is found, then v is dropped and the same procedure is carried out for G''. On the other hand, if no such dense subgraph is found, then v must definitely belong to every minimal dense subgraph inside G', hence add it to a set M of such vertices, and return M whenever it induces a dense subgraph of G. This entire procedure takes $O(|V|^2(|V| + |E|))$ (in practice $O(|V||E|)$) steps, where V and E respectively are the sets of vertices and edges of G', which in the worst case could be the original constraint graph G.

2.2.1 Simplifying found clusters

Once a minimal dense subgraph S_i is located in G_i, Step 2 of the recursive DR-planning process is to simplify it, thereby transforming the constraint graph G_i at the previous stage, 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, so that an optimal DR-plan can be found. Intuitively, there are problems with a simplistic approach such as condensing the cluster found in G_i into a single vertex in G_{i+1} with the appropriate degree of freedom: The goal is to minimize the information lost during the simplification. The cluster corresponding to the minimal dense subgraph interacts with the rest of the constraint graph, using its *frontier vertices,*. i.e, those vertices that are connected to the outside of the cluster. The goal of the Frontier vertex algorithm (FA) is to preserve the frontier vertex information, so that an optimal DR-plan can be found.

After a nontrivial, minimal or extended dense subgraph S_i of G_i is discovered, the subgraph induced by its internal vertices (those vertices that are not connected by an edge to the vertices outside S_i) is contracted in to one vertex (the core vertex) c_i. However, this core vertex is connected to each frontier vertex v of S_i by a combined edge whose weight is the the sum of the weights of the original edges connecting internal vertices to v. *The frontier vertices, edges connecting them, and their weights remain unchanged.* The weight of the core vertex is chosen so that the density of the entire simplified cluster is exactly equal $-D$ where D is the geometry-dependent constant. This process of finding solvable S_i and simplifying them is repeated, until the solvable S_m found it the entire remaining graph G_m.

See Figure 6 for an illustration of how FA constructs various clusters in the final DR-plan of Figure 5. As noted before, although it is most convenient to describe FA as a linear sequence of simplification steps, in fact, FA constructs a partially ordered DR-plan.

The formal graph transformation performed by FA is described in [13, 14], using so-called simplifier maps which permit the proof of various formal properties of FA, which in turn, illustrate FA's superior performance with respect to formal measures developed for DR-planners in [12]; a sketch of the FA implementation is also presented there. Many of the actual algorithms, datastructures and various crucial issues such as dealing with trivial, rotationallly symmetric, externally overconstrained clusters are worked out in [28, 29], which describe

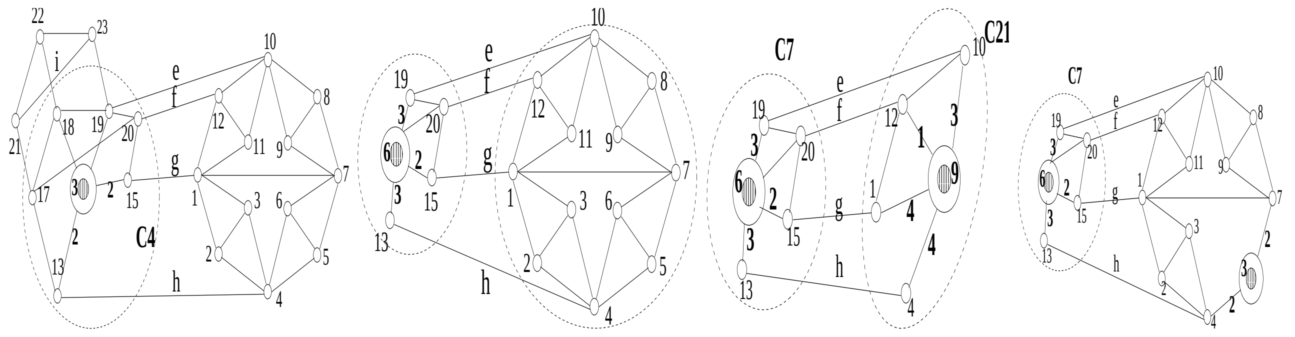


Figure 6: From left: FA's simplification of graph according to DR-plan in Figure 5; clusters are simplified in their numbered order: C4 is simplified before C7 etc.

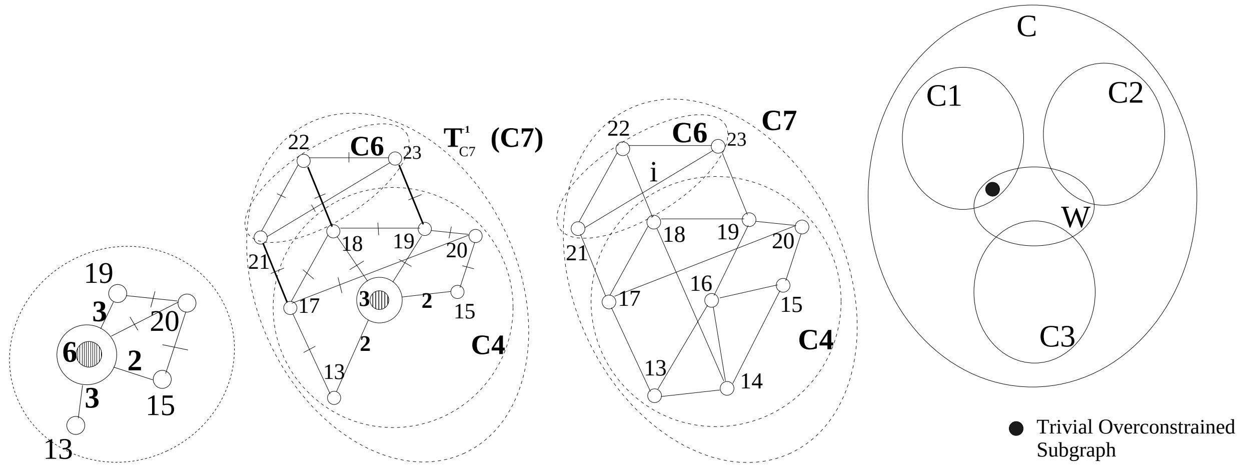


Figure 7: From left. Cluster C_7 of Figure 6. C_7 before transformation: $T_{C_7}^{-1}(C_7)$ – bold edges are relevant edges used for resolving C_7 and both bold and marked edges are unchanged edges. Underlying subgraph G_{C_7} corresponding to C_7 (from graph in Figure 5). Third key property of FA - W must be a child of C

the entire FRONTIER system [30]. In the next subsection, we list some of the key properties of FA that we repeatedly use in this paper.

2.2.2 Key properties of FA

The correctness of FA follows from the **first** property of FA that each internal node in the FA DR-plan represents a well-constrained *cluster* C (i.e, a simplified representation of the underlying nontrivial wellconstrained subgraph G_C of G), obtained (in the second step above) by applying a simplifying transformation T_C to a well-constrained subgraph consisting of child clusters of C. This subgraph is and is often also denoted $T_C^{-1}(C)$. This subgraph is isolated as S_i in the decomposition step of some i^{th} iteration of FA (recall that FA's iterations impose a linear order consistent with the partial order of the DR-plan). See Figure 7.

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 (as do most DR-plans) have the property that if the only changes made to G are within a cluster C and and these preserve the well-constrainedness of C then they will preserve the well-constrainedness of the ancestor clusters of C as well.

A **second** key property of FA is that the Frontier vertices and edges of the clusters C are *unchanged*, i.e, they 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 *unchanged edges* (see Figure 7) that are in 1-1 correspondence with edges from the original graph G. These represent constraints that are used in resolving or

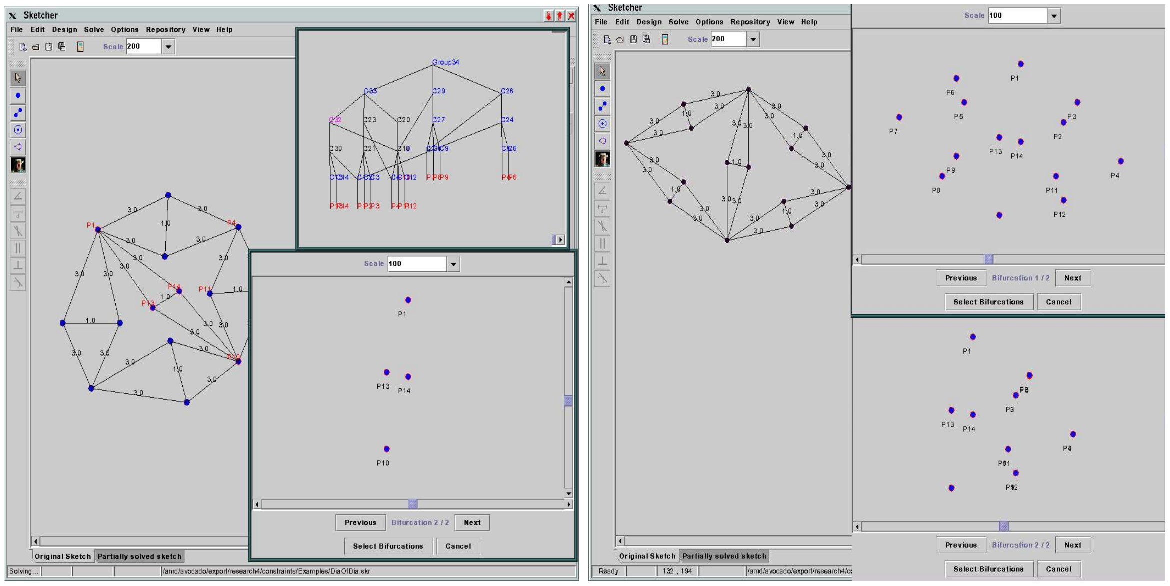

Sketcher
File Edit Design Solve Options Repository View Help
Scale 200
Scale 100
Previous Bifurcation 2 / 2 Next
Select Bifurcations Cancel
Previous Bifurcation 1 / 2 Next
Original Sketch Partially solved sketch
Solving...
Ready
P1
P13
P14
P10
Group34

Figure 8: Screenshots from FRONTIER: (Left) 2d sketch, DR-plan and 1 solution possibility for a lower level cluster subsystem; (Right)Root cluster of DR-plan being resolved after choosing solution for a child cluster on left figure

recombining 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 8 is a FRONTIER screencapture 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 graph of Figure 3 and Figure 8, 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.

A **third** property (crucial for FA's correctness) concerns the structure of a cluster C before it is simplified, i.e, the structure of $T_C^{-1}(C)$. First, C satisfies a *cluster minimality* in the following sense: no proper subset of C's or 2 or more child clusters C_i induce a dense proper subgraph of G. Second, if G_C (the original subgraph underlying C) is nontrivial and contains a nontrivial well or well-overconstrained subgraph W whose intersection with any one of the child clusters C_i of C in $T_C^{-1}(C)$, is a trivial, overconstrained subgraph (e.g, a single point as shared object– see Figure 7), then W must itself have been represented as a child cluster C_i in $T_C^{-1}(C_i)$.

FA's design objective that led to preserving frontier vertex information was the generation optimal DR-plans, i.e, those for which the maximal fan-in (number of child clusters C_i that combine into any parent cluster C) is minimized. This trades off, however, with the number of clusters in the DR-plan. Similarly, maintaining the property of the previous paragraph additionally has the effect of increasing the number of clusters in the DR-plan. A **fourth** property of FA that is crucial to prevent an exponential mushrooming of clusters and for maintaining polynomial complexity of FA is that two clusters C_i of the DR-plan do not overlap on non-trivial well-(over)constrained graphs, unless one actually contains the other. By FA's simplification procedure, this property holds for their corresponding subgraphs G_{C_i} as well. Due to this property, 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, although, this typically takes only $O(|V||E|)$ steps. In any case, the total complexity of the FA DR-planner in the 2d case is bounded by $O(|V|^4(|V|+|E|))$ steps; in practice it typically takes $O(|V|^2|E|)$ steps.

A **fifth** property of the FA DR-plan is used crucially here. If G is overconstrained, then for any nontrivial well-overconstrained subgraph W, we can read off from the DR-plan $D(G)$ the unique minimal nontrivial cluster C, whose corresponding subgraph G_C contains W. However, G_C could be significantly larger than W. The uniqueness follows from the 4th property in the previous paragraph. If there is a unique, minimal, nontrivial, well-overconstrained graph W, then there is a *unique, minimal, nontrivial overconstrained cluster* C, in the DR-plan $D(G)$, whose corresponding subgraph G_C contains W. Here, minimality of C means that no descendant cluster's subgraph contains W. This cluster C is denoted $S(D(G))$ and can be read off from $D(G)$ as the unique minimal cluster C where $T_C^{-1}(C)$ is overconstrained. Since the child clusters C_i of C are (by the FA simplification) always represented as wellconstrained 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.

A **sixth** key property is that the FA DR-plan incorporates (is a proper refinement of) an input hierarchy of (well or well-overconstrained) features or parts. I.e, the set features appear as clusters in the output DR-plan).

2.3 An Initial Problem Statement and Solution

We are now ready to formally define the problem being considered in this paper. The *reducible edges* in a geometric constraint graph G are defined as edges that could have their weight reduced by 1 without making G underconstrained, if G was well-overconstrained before. If G was underconstrained before, then these are exactly the edges that belong to atleast one of the maximal well-overconstrained subgraphs of G. and could be weight-reduced by 1 without underconstraining any of them. By the cut-based definition of underconstrained graphs given earlier, the reducible edges do not belong to any minimal cut of weight D in G (assuming the minimal cut separates the graph into nontrivial parts). An overconstrained graph is *1-overconstrained*, if it (or each of its maximal well-overconstrained subgraphs) becomes well-constrained as soon as exactly one of the edges in its reducible set has its weight reduced by 1. We restrict 1-overconstrained graphs to be nontrivial. Any 1-overconstrained graph should contain a 1-well-overconstrained graph as a subgraph. We can now make an initial statement of the overconstraint problem discussed in the Introduction.

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

A solution to this problem immediately presents itself due to the following fact and since we already know a good algorithm for isolating minimal dense subgraphs(satisfying any chosen density lowerbound), from Section 2.2.

Fact 2.2 *(i) Any nontrivial 1-overconstrained graph G may have many (1-)overconstrained subgraphs, but has a unique minimal, nontrivial, 1-overconstrained subgraph U (i.e, no proper subgraph of U is 1-overconstrained).*

(ii) By the minimality and uniqueness of U, it follows that an edge is in U if and only if every (1-)overconstrained subgraph of G contains it.

(iii) Furthermore it follows from Fact 2.1 that U is in fact 1-well-overconstrained.

(iv) Most importantly, U consists of exactly the reducible edges of G.

Proof: The uniqueness of U is proved as follows. Suppose to the contrary there were 2 minimal, nontrivial 1-overconstrained subgraphs U. Then by the inclusion-exclusion principle, unless their intersection is 1-overconstrained, their union has density atleast $D + 2$, i.e, atleast 1 more than that of any 1-overconstrained graph, which contradicts G's 1-overconstrainedness. On the other hand, if their intersection is 1-overconstrained, that contradicts the minimality of U_1 and U_2.

To show that the minimal 1-overconstrained graph U is exactly $L(G)$, assume not. First assume one of the edges outside U is reducible. Consider reducing the weight of one of the edges outside U by 1. After this reduction, G is still (barely) dense, i.e, has density $-D$, but it has a subgraph U of density $-D + 1$, and hence by Fact 2.1, the new G must be underconstrained. Next assume that one of the edges in U is not reducible. I.e, its weight reduction causes the reduced G to become underconstrained. However, its weight reduction causes U and all other 1-overconstrained subgraphs of G to be no longer 1-overconstrained, since they all contain U. But the reduced G's density is still $-D$. This contradicts Fact 2.1, which states that there should still be a nontrivial, 1-overconstrained subgraph W remaining in the now underconstrained G. $\square$

Initial Algorithmic Solution

Due to Fact 2.2, we can immediately obtain an efficient $O(|V|^2(|V|+|E|))$ (in practice $O(|V||E|)$) network flow based algorithm for this problem, by adapting the minimal dense subgraph location algorithm given in Section 2.2 (i.e, by simply increasing the density lower bound). This works for completely general geometric constrait hypergraphs in arbitrary dimensions. Once this graph is found, simply locate a minimal 1-overconstrained subgraph U in it by throwing out vertices one by one and trying to find a subgraph of density $-D+1$ within the resulting subgraph, exactly as in the minimal dense algorithm described in Section 2.2. The subgraph U thus found gives exactly $L(G)$ by Fact 2.2.

2.4 Drawbacks and Modified Problem Statement

The problem statement and solution given above are unsatisfactory since they do not satisfy the following requirements discussed informally in the Introduction. These directly define the modified problem.

Requirement 1: given an existing DR-plan $D(G)$, and given $D(G)$ as input, the algorithm should be significantly more efficient than the one given above: in particular, the list $L(G)$ of reducible edges of G should be automatically read off from the DR-plan, provided a small amount of additional information is stored and maintained along with the clusters (nodes) of the DR-plan.

Requirement 2: We would like the list of reducible edges to be output *incrementally* as sublists, in time typically linear in the number of *output* edges. This is acheivable by walking down the DR-plan, cluster by cluster, starting from the overconstrained cluster $S(D(G))$ that is read off from $D(G)$ as explained in Section 2.2.2. The first sublist is the set of reducible edges in G (if any) that are present as unchanged edges of G, in the cluster $S(D(G))$, i.e, these are the constraints that are resolved in the process of recombining the already solved child clusters of $S(D(G))$. See Section 2.2.2. The next sublists output are the sets of reducible edges in G (if any) 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 corresponding to A will be output earlier than the sublist corresponding to B. In other words, assuming all clusters in $D(G)$ have been solved, the edges or constraints in $L(G)$ are output in the *reverse* order in which they would be solved: "latest-solved-first." For a specified descendant cluster C, the sublist of reducible edges are among the relevant edges of C and these sublists would be output in a latest-solved-first order as we walk down the DR-plan.

Requirement 3: We would like the sublist of reducible edges for a particular (cluster or feature) C to be output on demand, preferably in a manner that inspects only the clusters on the path in the DR-plan from $S(D(G))$ to C (the clusters that contain C). This is meaningful for instance in the case of FA DR-plans, because they incorporate a designer's conceptual feature hierarchy – the features appear as clusters in the DR-plan.

Requirement 4: This concerns dynamic maintanence of the DR-plan. We would like the DR-plan to be easily and efficiently updated when one of reducible edges is actually reduced. In particular, we would like the optimal reorganization of the DR plan without reorganizing any of the original clusters that were preserved after 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 the bulk of 2d constraint systems, it is sufficient to use the triangle decomposition algorithm of [5] 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-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 [5], overconstrained problems are detected by two clusters sharing more than one geometric element, so that the 1-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 (removable) edges.

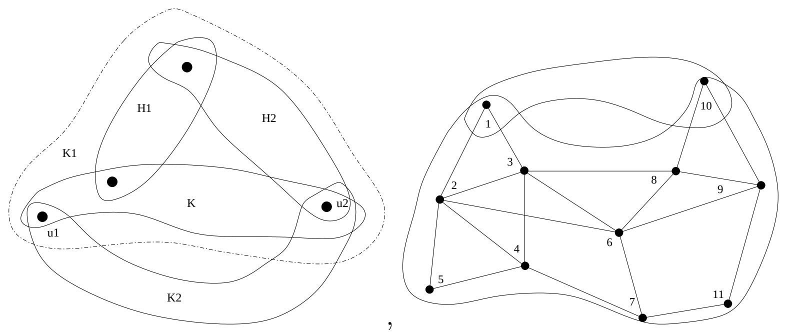


Figure 9: Left: Cluster pruning does not necessarily yield the critical subproblem. Right: Clusters H_1 and H_2 can be dropped from consideration.

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 9 (left). It is clear that if the clusters H_1 and H_2 can be removed, and that the resulting, smaller cluster K remains 1-overconstrained. Moreover, since both H_1 and H_2 must be overconstrained, 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 [12, 13], constraint-graph decomposition is not deterministic, although 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 9 (right).

What is needed is a tree re-ordering that exhibits the remaining redundancies. The decomposition method of cutting [36] achieves this, as we explain.

Let $G = (V, E)$ be a constraint graph. 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. Clearly, cluster pruning can be considered a general form of the 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.

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, a different approach achieves 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 a set S of constraint edges that are relevant to the 1-overconstrained situation. S 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 S 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 or removable constraints that contribute to the overconstrained situation. Using reference count, we can implement the test of remaining marked vertices in constant time, so that the algorithm overall is linear-time.

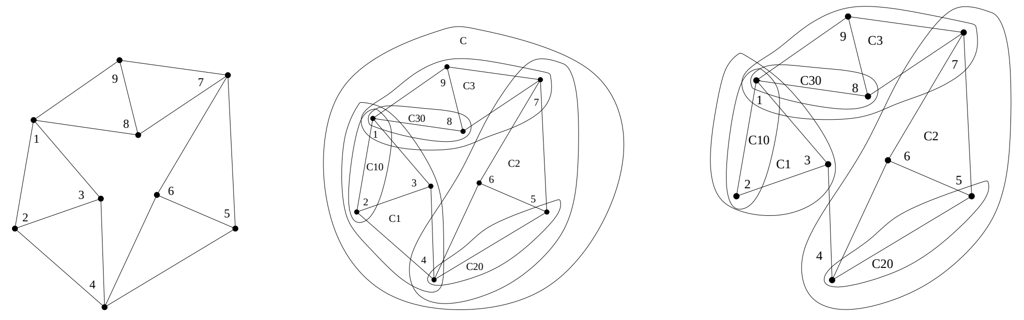


Figure 10: Left: A constraint graph Middle: its decomposition structure; Right: The result of edge reduction

Example. Consider the cluster shown in Figure 9(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 L and mark vertices 8 and 9. Examining 9 next, we add edges (8,9) and (6,9) to L 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.

Note: An additional difficulty arises from the fact that triangle decomposition cannot decompose all 2d well-constrained problems. In the case of solvers based on triangle decomposition, 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.

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 10 (left) is a constraint graph G and Figure 10 (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 10 (right) shows the result of the removing operation.

4 The Algorithm for general DR plans

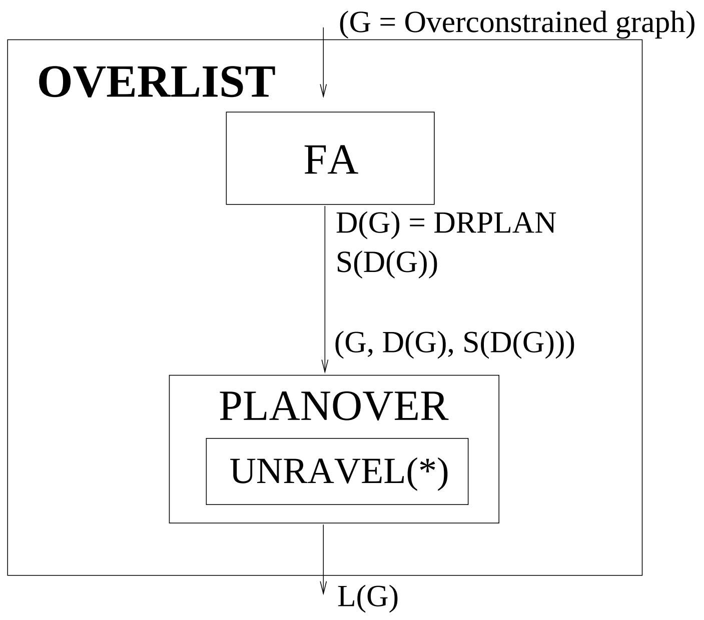
We now describe the algorithm Overlist, which takes as input a general geometric constraint hypergraph G that represents a geometric constraint system in 3d or arbitrary dimension, and is *known to be* 1-overconstrained. Therefore as in the previous section, $|E| = O(|V|)$. The algorithm uses FA to generate a DR plan $D(G)$ and and outputs the complete list $L(G)$ of reducible edges in G incrementally with and flexibly with respect to $D(G)$, in such a way as to satisfy all of the requirements of the modified problem 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. The algorithms are relatively simple, but their simplicity hides an intricate structure

Theorem 4.2 which establishes the algorithms correctness. Its proof is however relegated to the Appendix. The complexity analysis follows after a discussion of the importance of FA's key properties.

Algorithm OVERLIST uses a DR-plan $D(G)$ and the unique smallest (but nontrivial) 1-well-overconstrained subgraph corresponding to a cluster $S(D(G))$ – i.e, that is actually present in $D(G)$: both of these are output by the FA DR-planner described in Section 2.2 and Section 2.2.2 . As pointed out in Section 2.2.2, this subgraph corresponding to $S(D(G))$ may be quite a bit larger than the unique minimal 1-well-overconstrained subgraph of G which, in general, may not appear as a cluster in $D(G)$. The tuple $(G, D(G), S(D(G)))$ form the input to the algorithm Planover which is the main technical contribution here. 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 11 for the overall control flow of these algorithms.

```
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 $L_C$.
$P^*_C = P_C \cup P$
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 $T_C^{-1}(C)$ 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 results in Sections 2.2, 2.3, and 2.4
– 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* – see below
– UNRAVEL*, is never called on clusters C corresponding to singleton vertices of G; Moreover, the contact points in P_C are necessarily Frontier vertices of the C_i's and are hence unchanged vertices from 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 2.2.2 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



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 11: Overall control flow of algorithms; Text: differences in the example variants $G1$, $G2$, $G3$, $G4$ of Figure 12, and Figures 13, 14, 15, 16
.

– 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 Immediate Adaptations of the Algorithm

A key property of the above algorithm is that it adapts easily and *efficiently* to answering the following types of user queries (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 $D(G)$ – give 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 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 the first type of query above, first locate the unique relevant cluster C for e. Now, for both types of queries, the algorithm immediately adapts to give an efficient solution that requires *only* looking at the path of clusters from $S(D(G))$ to C in the DR-plan $D(G)$ (called the *unravelling path*): check whether UNRAVEL gets called on all of these clusters starting from the top of the path and by following the contact points. The rest of the graph can be completely ignored. Finally check whether UNRAVEL* gets called at C. If yes, then the relevant edges are C are reducible edges that would be output as L_C by Algorithm PLANOVER above.

More significantly, these algorithms are easily implemented on top of the DR-planners of existing constraint solvers. This is because the entire information flow in the above algorithms involves simple pieces of information that can be stored and and updated routinely as part of the DR-plan: 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

We use the realistic 2d constraint system of Figure 5 to illustrate the working of Algorithm OVERLIST in Figure 12. Figures 13, 14, 15 and 16 all further illustrate the key points of the algorithm using variants of the first example. These examples exhibit the complexity of geometric constraint systems that occur in practice and which require the general decomposition capabilities of our FA DR-planner, and of the algorithms given in this section. As mentioned before, the algorithms given in this section are completely general and work for constraint hypergraphs obtained from 3d constraint systems.

In the case of the graph $G2$ 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 2.2.2. 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 2.2.2. 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 2.2.2 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.

A Note on Implementation

The FA DR-plans for the examples given here and many other realistic 2d and 2d 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 [30], where the source code can also be downloaded. The algorithms described here will be incorporated into FRONTIER's "update" mode.

Importance of the FA DR-plan's properties

The correctness of Algorithms OVERLIST and PLANOVER rely crucially on the properties of the FA DR-plan given in Section 2.2.2. This is elucidated by the control flow and the text in Figure 11, which distinguishes between the variants in Figures 13, 14, 15 and 16.

4.3 Correctness and complexity

The relatively simple description of the algorithm hides an intricate proof of correctness. 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. *The proof is given in the Appendix.*

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, if it not in any of the sublists L_C's output) is not reducible in G.*

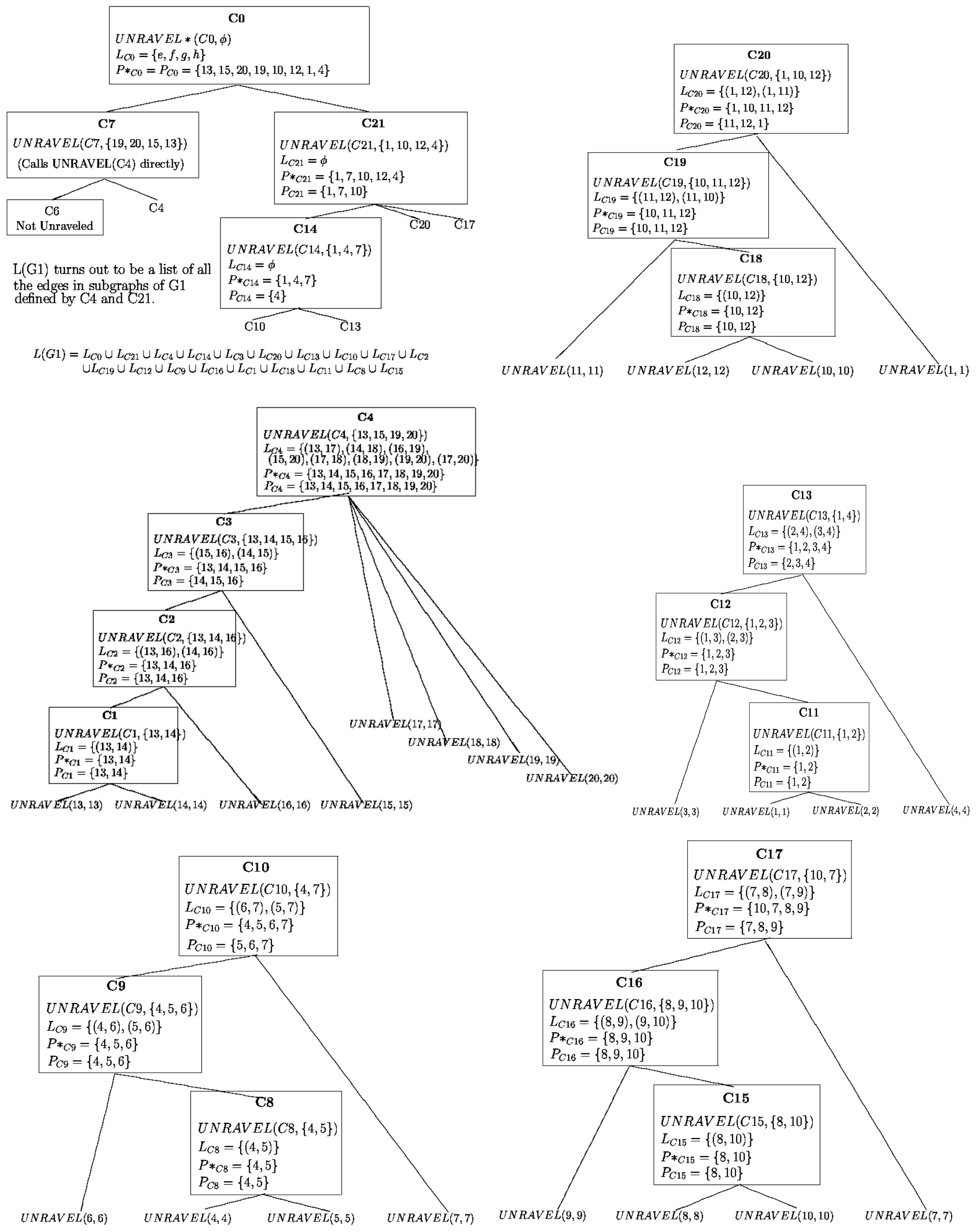
4.3.1 Complexity

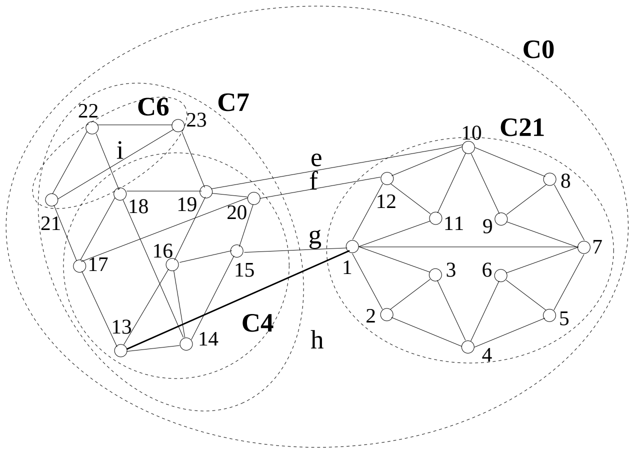
The complexity of constructing the FA DR-plan $D(G)$ for G is given in Section 2.2.2 and by Figure 11, it gets included into the complexity of Algorithm OVERLIST: it represents the dominant complexity term.

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, the depth of C in the DR-tree, and the number of output edges or constraints?.*

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 $T_C^{-1}(C)$ 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.

Note: Our complexity analysis assumes that t is a linear function – simple graph datastructures ensure this, even if contact points and relevant edges have to be computed from scratch *without* being maintained as part of


C0
$UNRAVEL*(C0, \phi)$
$L_{C0} = \{e, f, g, h\}$
$P*_{C0} = P_{C0} = \{13, 15, 20, 19, 10, 12, 1, 4\}$
C7
$UNRAVEL(C7, \{19, 20, 15, 13\})$
(Calls UNRAVEL(C4) directly)
C6
Not Unraveled
C4
C21
$UNRAVEL(C21, \{1, 10, 12, 4\})$
$L_{C21} = \phi$
$P*_{C21} = \{1, 7, 10, 12, 4\}$
$P_{C21} = \{1, 7, 10\}$
C20
C17
C14
$UNRAVEL(C14, \{1, 4, 7\})$
$L_{C14} = \phi$
$P*_{C14} = \{1, 4, 7\}$
$P_{C14} = \{4\}$
C10
C13
L(G1) turns out to be a list of all the edges in subgraphs of G1 defined by C4 and C21.
$L(G1) = L_{C0} \cup L_{C21} \cup L_{C4} \cup L_{C14} \cup L_{C3} \cup L_{C20} \cup L_{C13} \cup L_{C10} \cup L_{C17} \cup L_{C2} \cup L_{C19} \cup L_{C12} \cup L_{C9} \cup L_{C16} \cup L_{C1} \cup L_{C18} \cup L_{C11} \cup L_{C8} \cup L_{C15}$
C20
$UNRAVEL(C20, \{1, 10, 12\})$
$L_{C20} = \{(1, 12), (1, 11)\}$
$P*_{C20} = \{1, 10, 11, 12\}$
$P_{C20} = \{11, 12, 1\}$
C19
$UNRAVEL(C19, \{10, 11, 12\})$
$L_{C19} = \{(11, 12), (11, 10)\}$
$P*_{C19} = \{10, 11, 12\}$
$P_{C19} = \{10, 11, 12\}$
C18
$UNRAVEL(C18, \{10, 12\})$
$L_{C18} = \{(10, 12)\}$
$P*_{C18} = \{10, 12\}$
$P_{C18} = \{10, 12\}$
$UNRAVEL(11, 11)$
$UNRAVEL(12, 12)$
$UNRAVEL(10, 10)$
$UNRAVEL(1, 1)$
C4
$UNRAVEL(C4, \{13, 15, 19, 20\})$
$L_{C4} = \{(13, 17), (14, 18), (16, 19), (15, 20), (17, 18), (18, 19), (19, 20), (17, 20)\}$
$P*_{C4} = \{13, 14, 15, 16, 17, 18, 19, 20\}$
$P_{C4} = \{13, 14, 15, 16, 17, 18, 19, 20\}$
C3
$UNRAVEL(C3, \{13, 14, 15, 16\})$
$L_{C3} = \{(15, 16), (14, 15)\}$
$P*_{C3} = \{13, 14, 15, 16\}$
$P_{C3} = \{14, 15, 16\}$
C2
$UNRAVEL(C2, \{13, 14, 16\})$
$L_{C2} = \{(13, 16), (14, 16)\}$
$P*_{C2} = \{13, 14, 16\}$
$P_{C2} = \{13, 14, 16\}$
C1
$UNRAVEL(C1, \{13, 14\})$
$L_{C1} = \{(13, 14)\}$
$P*_{C1} = \{13, 14\}$
$P_{C1} = \{13, 14\}$
$UNRAVEL(13, 13)$
$UNRAVEL(14, 14)$
$UNRAVEL(16, 16)$
$UNRAVEL(15, 15)$
$UNRAVEL(17, 17)$
$UNRAVEL(18, 18)$
$UNRAVEL(19, 19)$
$UNRAVEL(20, 20)$
C13
$UNRAVEL(C13, \{1, 4\})$
$L_{C13} = \{(2, 4), (3, 4)\}$
$P*_{C13} = \{1, 2, 3, 4\}$
$P_{C13} = \{2, 3, 4\}$
C12
$UNRAVEL(C12, \{1, 2, 3\})$
$L_{C12} = \{(1, 3), (2, 3)\}$
$P*_{C12} = \{1, 2, 3\}$
$P_{C12} = \{1, 2, 3\}$
C11
$UNRAVEL(C11, \{1, 2\})$
$L_{C11} = \{(1, 2)\}$
$P*_{C11} = \{1, 2\}$
$P_{C11} = \{1, 2\}$
$UNRAVEL(3, 3)$
$UNRAVEL(1, 1)$
$UNRAVEL(2, 2)$
$UNRAVEL(4, 4)$
C10
$UNRAVEL(C10, \{4, 7\})$
$L_{C10} = \{(6, 7), (5, 7)\}$
$P*_{C10} = \{4, 5, 6, 7\}$
$P_{C10} = \{5, 6, 7\}$
C9
$UNRAVEL(C9, \{4, 5, 6\})$
$L_{C9} = \{(4, 6), (5, 6)\}$
$P*_{C9} = \{4, 5, 6\}$
$P_{C9} = \{4, 5, 6\}$
C8
$UNRAVEL(C8, \{4, 5\})$
$L_{C8} = \{(4, 5)\}$
$P*_{C8} = \{4, 5\}$
$P_{C8} = \{4, 5\}$
$UNRAVEL(6, 6)$
$UNRAVEL(4, 4)$
$UNRAVEL(5, 5)$
$UNRAVEL(7, 7)$
C17
$UNRAVEL(C17, \{10, 7\})$
$L_{C17} = \{(7, 8), (7, 9)\}$
$P*_{C17} = \{10, 7, 8, 9\}$
$P_{C17} = \{7, 8, 9\}$
C16
$UNRAVEL(C16, \{8, 9, 10\})$
$L_{C16} = \{(8, 9), (9, 10)\}$
$P*_{C16} = \{8, 9, 10\}$
$P_{C16} = \{8, 9, 10\}$
C15
$UNRAVEL(C15, \{8, 10\})$
$L_{C15} = \{(8, 10)\}$
$P*_{C15} = \{8, 10\}$
$P_{C15} = \{8, 10\}$
$UNRAVEL(9, 9)$
$UNRAVEL(8, 8)$
$UNRAVEL(10, 10)$
$UNRAVEL(7, 7)$
Figure 12: Working of OVERLIST showing calls to UNRAVEL for the graph $G1$ and DR-plan of Figure 5;



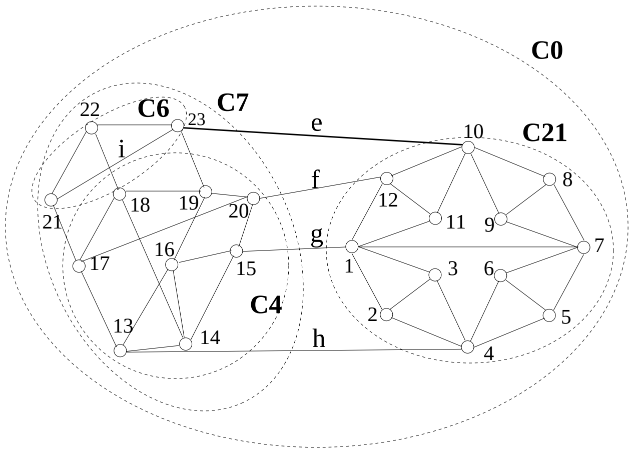
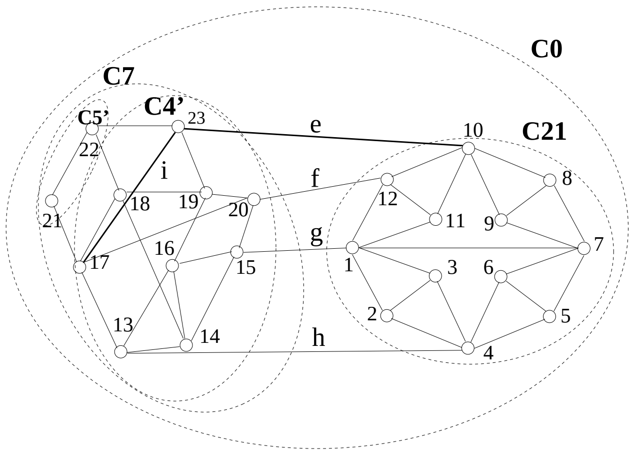


Figure 13: Left - Graph $G2$: edge h changes from $G1$; Right - Graph $G3$ edge e changes from $G1$; in both cases, no change in FA DR-plan



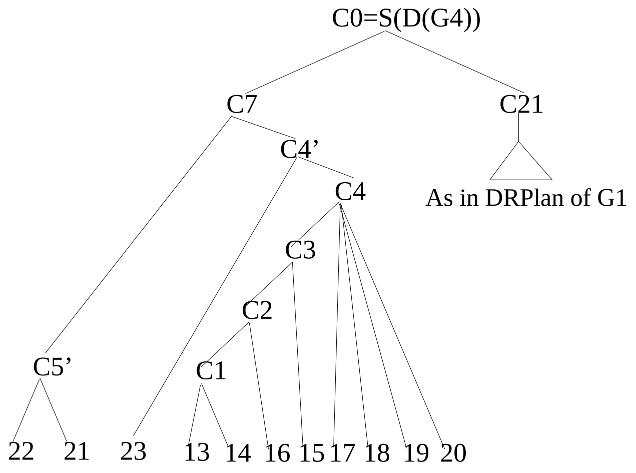
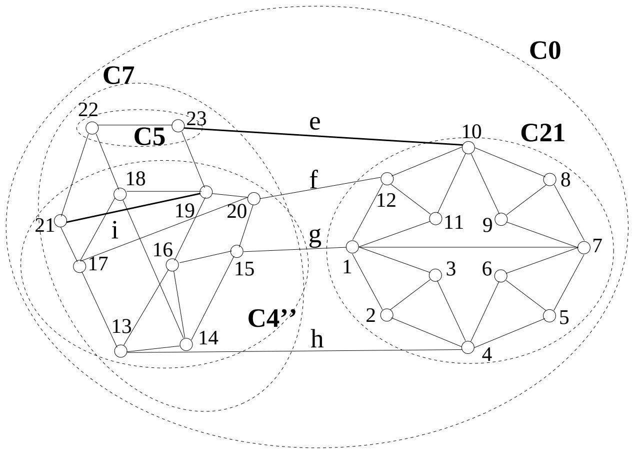


Figure 14: Graph $G4$: edge e and i change from $G1$; Right: changed DR-plan



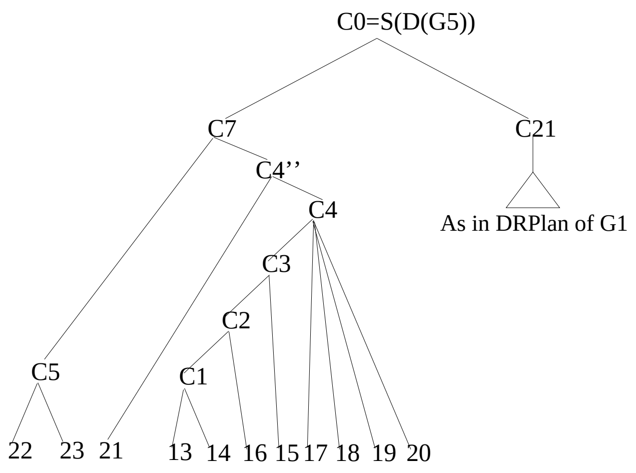


Figure 15: Graph $G5$: edge e and i change from $G1$; Right: changed DR-plan

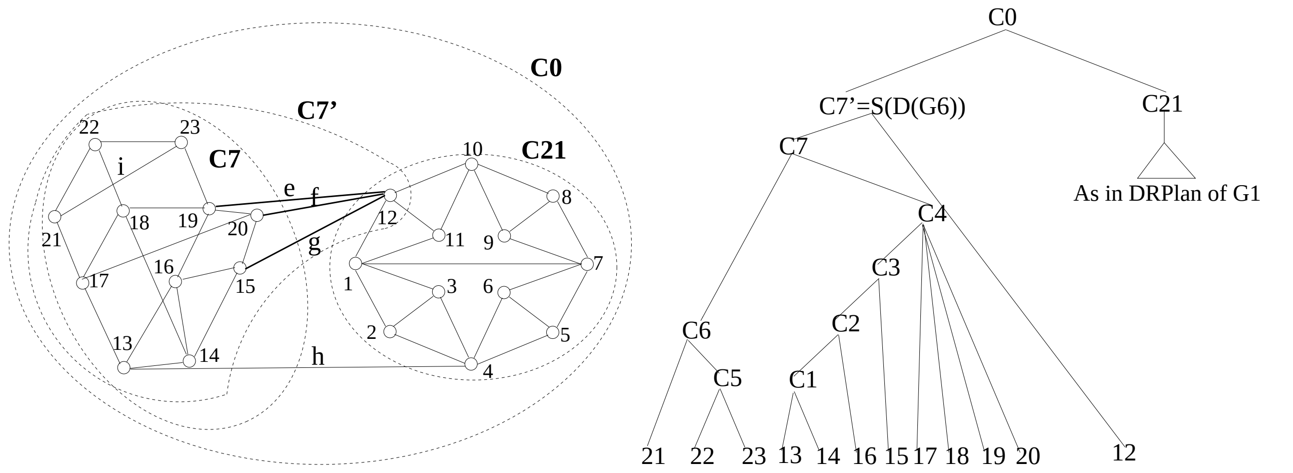


Figure 16: Graph $G6$: edge f and g changed from $G1$: Right: changed DR-plan as well as input cluster $S(D(G))$ for PLANOVER

the DR-plan. If these data are maintained systematically using high-efficiency datastructures, 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 (usually 2), 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 2.2.2). 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 $O(l_C(|V|+|E|))$. Here $|E|$ can be assumed to be the number of *output* edges since none others are inspected. 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 the number of output edges $|E|$, 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 2.2.2). 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 $O(|E|)$, i.e, it is linear in the number of output edges.

5 Updating the DR-plan after edge reduction

We now briefly consider 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 that is near-optimal, but preferably without reorganizing any of the (maximal) descendant clusters F of $S(D(G))$ whose subgraphs G_F remained wellconstrained after the reduction (recall the definition of optimal in Section 2.1). This can be done by a simple use of 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, these can easily be maintained 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 17.

When one of the edges in L_C is weight-reduced by 1, the clusters that are destroyed i.e 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 subDR-plans are rooted at the clusters F which are exactly the children of the clusters E, excepting the children that are directly on p. See Figure 17.

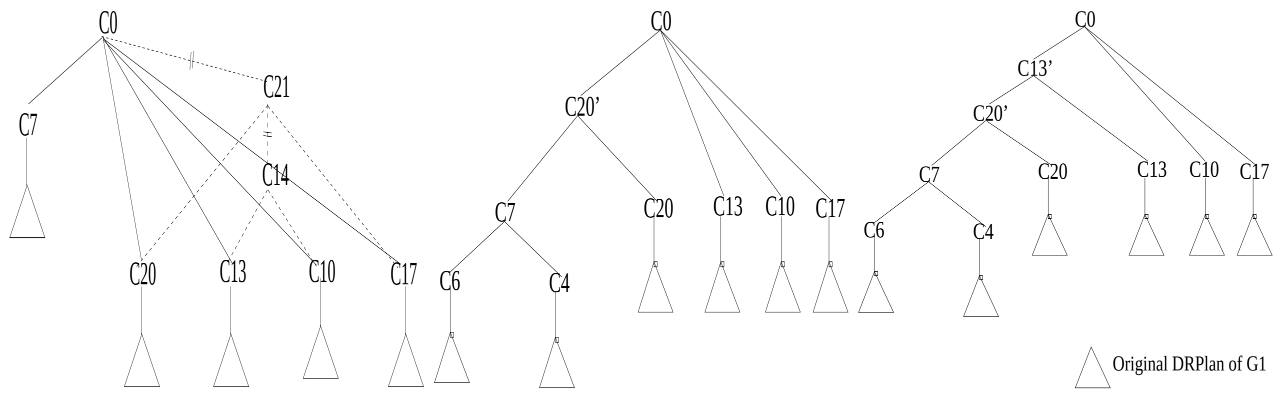


Figure 17: Left: unoptimized DR-plan after reduction of edge, old DR-plan is dotted; unravelling path is marked; Middle and Right: two optimization steps

An initial valid DR-plan attaches all of these 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 inorder to resolve or recombine $S(D(G))$ could be large, making the new DR-plan far from optimal.

However, the contact points inherited by the clusters F can be used in a straightforward manner to optimize this initial DR-plan. These contact points can be read off from the P_E^* for the ancestors E (of the clusters F) on the unravelling path p, including the cluster $S(D(G))$. These contact points should be considered in the 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 pictures below.

6 Conclusions and an Open Problem

A geometric constraint solver algorithm employs highly specialized mathematics and occupies a central position in many applications, for instance in computer-aided mechanical design. In most cases, the 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 those constraints may be essential for the application that should not be deleted. Thus, *all* relevant constraints must be identified. These must moreover be output efficiently and flexibly and on the user's demand, from desired subsystems of the constraint system which could represent specific parts or features. Furthermore, the algorithm should efficiently use information and data that are central to the constraint solver, and hence already exist, such as the DR-plan; these data must also be efficiently updated or dynamically maintained when one of the reducible constraints is removed by the user. We have formally identified these requirements, and given efficient algorithms that apply to fully general, 3d geometric constraint hypergraphs in arbitrary dimensions along with specific, simpler algorithms to deal with commonly occuring 2d cases. These algorithms will be incorporated into two established geometric constraint solvers Erep and Sketchworks at Purdue and FRONTIER [30] at Florida.

Note: We have only dealt with the case of 1-overconstrained problems. The general k-overconstrained case promises equal practical interest, but appears to be significantly more complex, based on preliminary work in [31].

Acknowledgements

Many thanks to Nandhini Kohareswaran for the pictures.

References

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

[2] 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.

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

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

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

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

[7] 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.

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

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

[10] 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.

[11] 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.

[12] 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.

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

[14] 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. .

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

[16] 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.

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

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

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

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

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

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

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

[24] 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.

[25] D-Cubed In *Commercial Constraint Solver*

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

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

[28] 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*

[29] 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*

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

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

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

[33] 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.

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

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

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

Appendix

Proof of Theorem 4.2: Item 1 holds immediately 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 2.2.2, 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 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. Any 1-well-overconstrained subgraph W of G_S:*

(i) 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; 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}'s.
(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 $C = S(D(G))$. □(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 atleast one of G_{C_i} on a trivial overconstrained subgraph. However, by a key property of FA DR-plans in Section 2.2.2, such a subgraph would have to directly yield or be represented as one of the child clusters of C, contradicting Claim 2a. □(Claim 3)

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

Next we show that the W_{C_i}'s 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 atleast $\sum_{i \in Q}$ density(G_{C_i})+ (total edge weight of L_C^Q) which is atleast $\sum_i$ density(W_{C_i})+ (total edge weight of L_C^Q) which is atleast 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 2.2.2 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 2.2.2, 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.
□(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 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 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 atleast 2 of C's children C_i. This is a different proof of Claim 2a of the base case. 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 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 atleast 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 2.2.2, 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 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 $T_C^{-1}(C)$. 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 18. 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 $T_{E_i}^{-1}(E_i)$ as well as the contact points in $P^*_{E_l}$

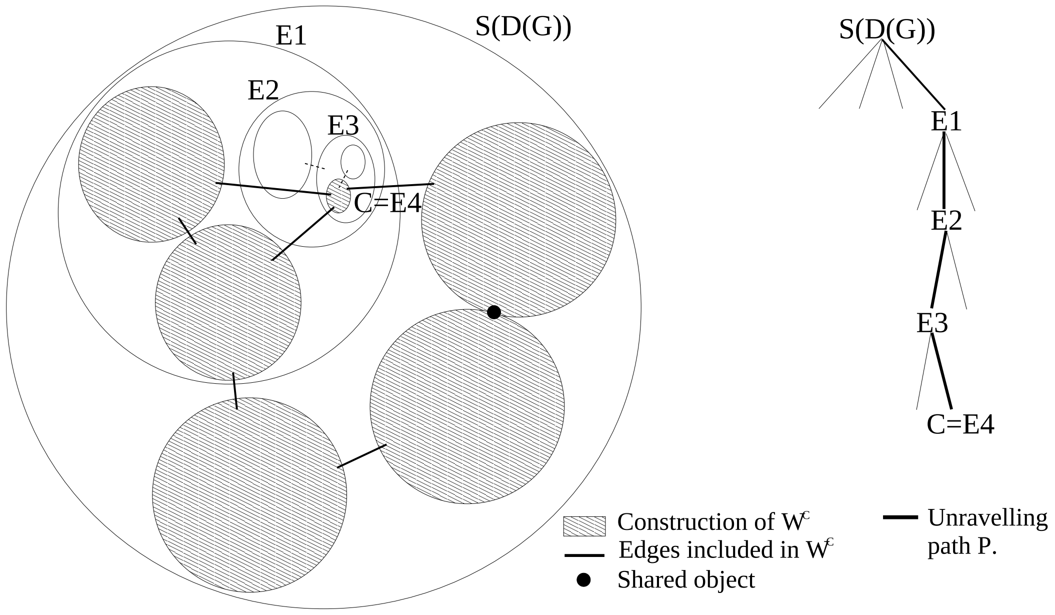


Figure 18: Construction of W^C for cluster C

(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 $T_C^{-1}(C)$.

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, we perform exactly this embedding by including all the edges in the well-constrained G_{C_i} into W^C. □(Item 3, and Theorem 4.2)