J. Symbolic Computation (2001) **31**, 367–408
doi:10.1006/jsco.2000.0402
Available online at http://www.idealibrary.com on IDEAL®

Decomposition Plans for Geometric Constraint Systems, Part I: Performance Measures for CAD

CHRISTOPH M. HOFFMAN[†§], ANDREW LOMONOSOV[‡¶]
AND MEERA SITHARAM[‡¶‖]

[†] *Computer Science, Purdue University, West Lafayette, IN 47907, U.S.A.*
[‡] *CISE, University of Florida, Gainesville, FL 32611-6120, U.S.A.*

A central issue in dealing with geometric constraint systems for CAD/CAM/CAE is the generation of an optimal decomposition *plan* that not only aids efficient solution, but also captures design intent and supports conceptual design. Though complex, this issue has evolved and crystallized over the past few years, permitting us to take the next important step: in this paper, we formalize, motivate and explain the decomposition–recombination (DR)-planning problem as well as several performance measures by which DR-planning algorithms can be analyzed and compared. These measures include: generality, validity, completeness, Church–Rosser property, complexity, best- and worst-choice approximation factors, (strict) solvability preservation, ability to deal with underconstrained systems, and ability to incorporate conceptual design decompositions specified by the designer. The problem and several of the performance measures are formally defined here for the first time—they closely reflect specific requirements of CAD/CAM applications.

The clear formulation of the problem and performance measures allow us to precisely analyze and compare existing DR-planners that use two well-known types of decomposition methods: SR (constraint shape recognition) and MM (generalized maximum matching) on constraint graphs. This analysis additionally serves to illustrate and provide intuitive substance to the newly formalized measures.

In Part II of this article, we use the new performance measures to guide the development of a new DR-planning algorithm which excels with respect to these performance measures.

1. Introduction and Motivation

Geometric constraints are at the heart of computer aided engineering applications (see, e.g. Hoffmann and Rossignac, 1996; Hoffmann, 1997), and also arise in many geometric modeling contexts such as virtual reality, robotics, molecular modeling, teaching geometry, etc. Both parts of this paper, however, deal with geometric constraint systems primarily within the context of product design and assembly. Figure 3 illustrates those (boldface) components within a standard CAD/CAM/CAE master model architecture (Bronsvoort and Jansen, 1994; Kraker *et al.*, 1997; Hoffmann and Joan-Arinyo, 1998) that Parts I and II of this paper directly address.

[§]Supported in part by NSF Grants CDA 92-23502 and CCR 95-05745, and by ONR Contract N00014-96-1-0635.
[¶]Supported in part by NSF Grant CCR 94-09809.
[‖]Corresponding Author: E-mail: `sitharam@cise.ufl.edu`

0747–7171/01/040367 + 42 $35.00/0

NOTE. Throughout this manuscript, *slant* is used for emphasis, while *italics* are used for introducing new terminology.

Informally, a *geometric constraint problem* 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 three dimensions. The constraints are spatial and include logical constraints such as incidence, tangency, perpendicularity, etc., and metric constraints such as distance, angle, radius, etc. The spatial constraints can usually be written as algebraic equations whose variables are the coordinates of the participating geometric objects.

A solution of a geometric constraint problem is a real zero of the corresponding algebraic system. In other words, a 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 thrree-dimensional space. For recent reviews of the extensive literature on geometric constraint solving see, e.g. Hoffmann *et al.* (1998), Kramer (1992) and Fudos (1995).

1.1. THE MAIN REASON TO DECOMPOSE GEOMETRIC CONSTRAINT SYSTEMS

Currently there is a lack of effective spatial variational constraint solvers and assembly constraint solvers that scale to large problem sizes and can be used interactively by the designer as *conceptual* tools throughout the design process. Almost all current CAD/CAM systems primarily use a non-variational, history-based three-dimensional constraint mechanism. This basic inadequacy in spatial constraint solving seriously hinders progress in the development of intelligent and agile CAD/CAM/CAE systems.

One governing issue is efficiency: computing the solution of the non-linear algebraic system that arises from geometric constraints is computationally challenging, and except for very simple geometric constraint systems, this problem is not tractable in practice without further machinery. The so-called constraint propagation-based solvers (e.g. Gao and Chou, 1998a; Klein, 1998) generally suffer from a drawback that cannot be easily overcome: they find it difficult to decompose cyclically constrained systems, an essential feature of variational problems. Direct approaches to algebraically processing the *entire* system include:

(1) standard methods for polynomial ideal membership and locating solutions in algebraically closed fields, for example using Gröbner bases (Ruiz and Ferreira, 1996) or the Wu–Ritt method (Chou *et al.*, 1996; Gao and Chou, 1998b);
(2) numerous algorithms and implementations for solving over the reals based on the methods of, for example, Canny (1993), Renegar (1992), Collins (1975) and Lazard (1981), etc.; and
(3) algorithms for decomposing and solving sparse polynomial systems based on Canny and Emiris (1993), Sturmfels (1993), Khovanskii (1978), Sridhar *et al.* (1993) and Sridhar *et al.* (1996), etc.

These direct algebraic solvers deal with general systems of polynomial equations, i.e. they do not exploit geometric domain knowledge; partly as a result, they have at least exponential time complexity and they are slow in practice as well. In addition, they do not

take into account design considerations and as a result, cannot assist in the conceptual design process. These drawbacks apply to direct numerical solvers as well, including those that use homotopy continuation methods, see, for example, Durand (1998). The problem would be further compounded if we allowed constraints that are natural in the design process, but which must be expressed as inequalities, such as "point P is to the left of the oriented line L in the plane". Such additions would necessitate using cylindrical algebraic decomposition-based techniques (Collins, 1975), such as Grigor'ev and Vorobjov (1988), Lazard (1991) and Wang (1993) which have a theoretical worst-case complexity of $O(2^{n^2})$, where n is the algebraic size of the problem; or alternatively require the use of non-linear optimization techniques, all of which are slow enough in practice that they do not represent a viable option for large problem sizes.

With regard to efficiency, the following rule of thumb has therefore emerged from years of experimentation with geometric, spatial constraint solvers in engineering design and assembly: the use of direct algebraic/numeric solvers for solving large subsystems renders a geometric constraint solver practically useless (see Durand, 1998, for a natural example of a geometric constraint system with six primitive geometric objects and 15 constraints, which has repeatedly defied attempts at tractable solution). The overwhelming cost in geometric constraint solving 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.

Therefore, the constraint solver should use geometric domain knowledge to develop a plan for decomposing the constraint system into small subsystems, whose solutions can be recombined by solving other small subsystems. The primary aim of this decomposition plan is to restrict the use of direct algebraic/numeric solvers to subsystems that are as small as possible. Hence the *optimal* or most efficient decomposition plan would minimize the size of the largest such subsystem. Any geometric constraint solver should first solve the problem of efficiently finding a close-to-optimal *decomposition–recombination (DR) plan*, because that dictates the usability of the 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.

In addition to optimality (efficiency), other equally important (and sometimes competing) issues arise from the fact that a DR-plan is a key conceptual component of the CAD model and should *aid* the overall design or assembly process. These issues will be discussed under "Desirable characteristics" later in this section.

A clean and precise formulation of the DR-planning problem is therefore a fundamental necessity. To our knowledge, despite its longstanding presence, the DR-planning problem has not yet been clearly isolated or precisely formulated, although there have been many prior, specialized DR-planners that utilize geometric domain knowledge (e.g. Serrano and Gossard, 1986; Crippen and Havel, 1988; Serrano, 1990; Havel, 1991; Owen, 1991, 1996; Kramer, 1992; Ait-Aoudia *et al.*, 1993; Pabon, 1993; Hoffmann and Vermeer, 1994, 1995; Bouma *et al.*, 1995; Hsu, 1996; Fudos and Hoffmann, 1996b, 1997; Latham and Middleditch, 1996). See Hoffmann *et al.* (1998) for an exposition of two primary classes of existing methods of decomposing geometric constraint systems; representative algorithms from these two classes are extensively analyzed in Section 4. In Part II of this paper we

describe and analyze a new algorithm that performs well with respect to the measures described below.

In the next two subsections we informally describe both the basic requirements of a DR-plan(ner) that dictate its overall structure, as well as desirable characteristics of the DR-plan(ner) that improve efficiency and assist in the design process.

1.2. BASIC REQUIREMENTS OF A DECOMPOSITION PLAN

Recall that a DR-plan specifies a plan for decomposing a constraint system into small subsystems and recombining solutions of these subsystems later. Therefore the first requirement of a DR-plan is that the solutions of the small subsystems in the decomposition can be recombined into a solution of the entire system. In other words, it should be possible to substitute the (set of) solution(s) of each subsystem into the entire system in a natural manner, resulting in a simpler system. Secondly, we would like these intermediate subsystems that are solved during the decomposition and recombination to be geometrically meaningful.

Together, these two requirements on the intermediate subsystems translate to a requirement that the subsystems be geometrically rigid. A *rigid or solvable* subsystem of the constraint system is one for which the set of real-zeroes of the corresponding algebraic equations is discrete (i.e. the corresponding real-algebraic variety is zero-dimensional), *after* the local coordinate system is fixed arbitrarily, i.e. after an appropriate number of degrees of freedom D are fixed arbitrarily. The constant D is usually the number of (translational and rotational) degrees of freedom available to any rigid object in the given geometry (three in two dimensions, six in three dimensions, etc.) and in some cases, D depends on other symmetries of the subsystem. An *underconstrained* system is not solvable, i.e. its set of real zeroes is not discrete (non-zero-dimensional). A *well-constrained* system is a solvable system where removal of any constraint results in an underconstrained system. An *overconstrained* system is a solvable system in which there is a constraint whose removal still leaves the system solvable. Solvable systems of equations are therefore well-constrained or overconstrained. That is, the constraints force a finite number of isolated real solutions, so one solution cannot be obtained by an infinitesimal perturbation of another. For example, Figure 1 shows a solvable subsystem of three points and three distances between pairs of points. A *consistently* overconstrained system is one which has at least one real zero.

NOTE. It is important to distinguish "solvable" from "has a real solution". Although (inconsistently) overconstrained (or even certain well-constrained) systems may have no real solutions at all, by our definition, since their set of real zeroes is discrete, they would still be considered "solvable". In general, whenever a subsystem of a constraint system is detected that has no real solutions, then the solution process would have to immediately halt and inform the designer of this fact.

Informally, a geometric constraint solver which solves a large constraint system E by using a DR-planner—to guide a direct algebraic/numeric solver capable of solving small subsystems—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

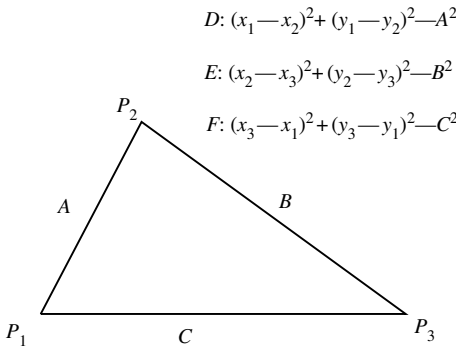

Figure 1. A solvable system of equations.

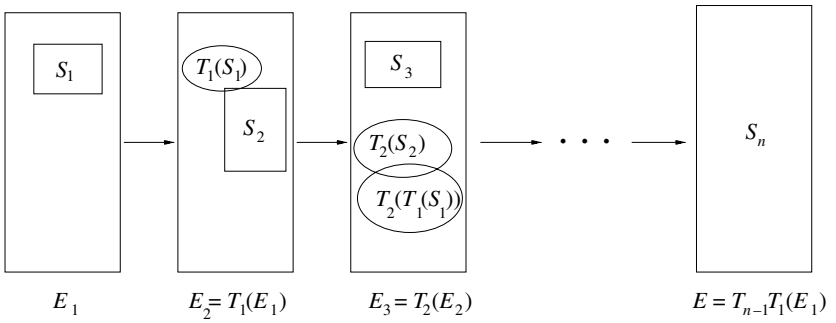

Figure 2. Solving a well-constrained system by decomposition: Step 1—rectangles, Step 3—ovals.

indicated by a rectangle in Figure 2. The subsystem S_i could be also chosen by the designer.

(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 2. 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.

A constraint solver that fits the above structural description—which we shall refer to as $\mathcal{S}$ in future discussions—is called a *decomposition–recombination (DR) solver*. (The formal definition imposes further requirements on simplifier T_i—see the next section.) This solver terminates when the small, solvable subsystem S_i found in Step 1 is the entire polynomial system E_i. An optimal DR-plan will minimize the size of the largest S_i. If the whole system is underconstrained, the solver should still solve its *maximal* solvable subsystems.

For the purpose of *planning* a solution sequence *a priori*, we would like to execute Steps 1 and 3 alone without access to the algebraic solver, and obtain a DR-plan, without actually solving the subsystems. That is, i.e. we would like the constraint solver to look as in Figure 3, with the DR-planner driving the direct algebraic/numeric solver.

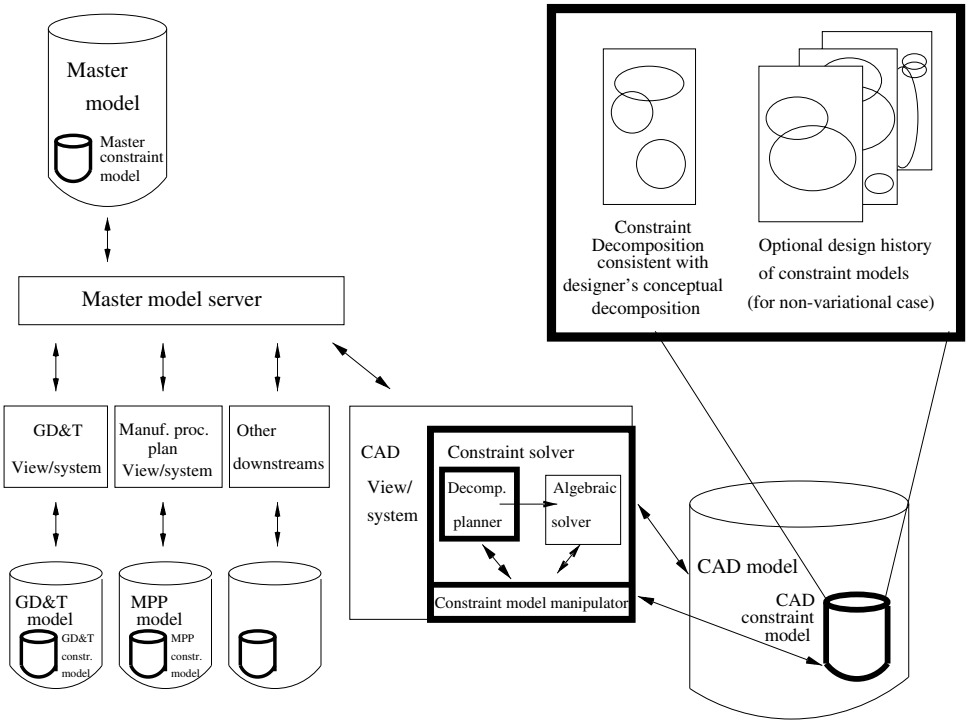

Figure 3. CAD/CAM/CAE master model architecture; this paper deals with boldface components.

To generate a DR-plan *a priori*, 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 solution to the S_i, and usually only depends on the degrees of freedom of the relevant geometric objects and geometric constraints.

1.3. desirable characteristics of DR-planners for CAD/CAM

We enumerate and informally describe a set $\mathcal{C}$ of natural characteristics desirable for a DR-planner. These will guide the formal definition of the DR-planning problem and the performance measures in the next section as well as the design of the new DR-planner in Part II of this paper. We begin with four criteria that directly follow from the overall structural description of a typical DR-planner $\mathcal{S}$ in the previous subsection.

(i) The DR-planner should be general, i.e. it should not be artificially restricted to a narrow class of decomposition plans; it should output a DR-plan if there is one, and it should be able to decompose underconstrained systems as well. Furthermore, if a DR-plan exists, the planner should run to completion irrespective of how and in what order the solvable subsystems S_i are chosen for the plan (a Church–Rosser property).

(ii) The planner should potentially output a close-to-optimal DR-plan (i.e. where the size of the largest solvable subsystem S_i is close-to-minimal). This dictates efficiency of solution of the constraint system.
(iii) The DR-planner should be fast and simple; the time complexity should be low, the planner should be fast in practice, easily implementable, and compatible with existing algebraic solvers, CAD systems, constraint models and manipulators.
(iv) The planner should utilize and take advantage of geometric domain knowledge, as well as other special properties of geometric constraints arising from the relevant design or assembly application or, in some situations, from a downstream manufacturing application.

Besides critically affecting the speed of constraint solving, a good DR-plan is a key component of the constraint model which participates in the overall process of design/assembly. This is especially so, since the constraint system is a component of the CAD model which the designer directly interacts with, and moreover, a DR-plan is nothing but a *hierarchical, structural* decomposition of the geometric constraint system. Therefore, maintaining a robust DR-plan—which reflects *design intent* at every level of refinement—is invaluable in improving efficiency, flexibility and transparency in the overall design process. These properties are crucial for intelligent CAD systems to facilitate designer interaction at a conceptual, early-design phase; in fact, an effective CAD system based on spatial, variational constraints to be effective must facilitate early-design interaction. This motivates adding the following desirable characteristics to the set $\mathcal{C}$.

(v) The DR-plan should be consistent with the design intent: in particular, the designer often has a multi-layered or hierarchical conceptual, design decomposition in mind, reflecting features or conglomerates of features in the case of product design and parts or subassemblies in the case of assembly. See e.g. Klein (1996), Mantyla *et al.* (1989), and Middleditch and Reade (1997). The designer would typically wish to further decompose the components of her design decomposition, as well as treat these components (recursively) as units that can be independently manipulated. For example, the geometric objects within a component are manipulated with respect to a local coordinate system, and a component as a whole unit is manipulated with respect to a global (next level) coordinate system. The DR-plan should therefore be a consistent extension and/or refinement of this conceptual design decomposition.
(vi) While the DR-plan is used to guide the algebraic solver, it should remain unaffected or adapt easily to the designer's intervention in the solution process, which is valuable for pruning combinatorial explosion: the designer can throw out meaningless or undesirable solutions of subsystems at an early stage. Such designer interference is also crucial for avoiding the use of inequality constraints: for example, instead of adding on a constraint that point P is to the left of line L, the designer can simply throw out any partial solutions that put P to the right of line L.
(vii) The DR-plan for solving a geometric constraint system should remain meaningful in the presence of constraints other than geometric constraints, such as equational constraints or parametric constraints expressing design intent.

Finally, the CAD system and the CAD model do not stand alone. In standard client–server based architectures (see, e.g. Bronsvoort and Jansen, 1994; Kraker *et al.*, 1997; Hoffmann and Joan-Arinyo, 1998), the CAD model is just one client's view of the product

master model (Newell and Evans, 1976; Semenkov, 1976), with which it has to be continually coordinated and made consistent. The master model in turn coordinates with other downstream production client systems which maintain other consistent views. These clients include geometric dimensioning and tolerancing systems (GD&T), and manufacturing process planners (MPP) for automatically controlled machining or assembly. Each client view contains potentially proprietary information that must be kept secure from the master model. Figure 3 illustrates this architecture and those parts that this paper directly deals with.

Each client view contains its own internal view of the constraint model and therefore coordination and consistency checks between the various views crucially involve, and vice versa may affect, the DR-plan. This leads to the following additions to the set $\mathcal{C}$ of desirable characteristics for DR-planners.

(viii) The DR-plan should be as robust as possible (see, e.g. Fang, 1992) to on-line changes made to the constraint system, and the DR-planner should be able to quickly alter the DR-plan in response to such changes. In particular, the DR-plan should ideally not change *at all* with numerical perturbations (within large ranges) to the constraints, thereby permitting the DR-plan to be computed as a pre-processing step. Addition, deletion, and modification of constraints and geometric objects to the constraint system occurs in numerous circumstances during the design process. For example:

(a) in the process of solving the system using the DR-plan, a subsystem S_i deemed solvable by a degree of freedom analysis may be found to have no real solutions, or may, in a degenerate case, turn out to be underconstrained or have infinitely many solutions, preventing a continuation of the solution process $\mathcal{S}$;

(b) in history-based CAD systems, the specification of the product takes the form of a progressive sequence of changes being made to successive partial specifications and the associated partial constraint systems;

(c) inferred or understood constraints are often added to an incomplete constraint specification at the discretion of the CAD system;

(d) in underconstrained situations, as might occur in designing flexible or moving parts and assemblies, the mapping of the configuration space of the resulting mechanism would require the repeated addition of constraints at the discretion of the constraint solver—the solutions to the resulting well-constrained systems would represent various configurations of the mechanism;

(e) for a variational spatial constraint solver to be effective or even usable, it would have to rely on extensive interactive constraint changes being made by the designer, sometimes in the course of solving; and

(f) finally, when one of the various clients in Figure 3 makes a change to its own view of the constraint model, this will result in consistency updates to the master constraint model which will, in turn, result in updates to the other views of the constraint model.

(ix) The DR-plan should isolate overconstrained subsystems which arise in assembly problems; furthermore, with multiple (possibly proprietary) views of the constraint model being kept by various clients as in Figure 3, constraint reconciliation often takes place and the DR-plan should facilitate this process, and viceversa, should be robust against this process. For a precise description of the constraint reconciliation problem, see Hoffmann and Joan-Arinyo (1998). The problem requires isolation

of overconstrained subsystems and is compounded in the case of non-variational, history-based design.

MAIN RESULTS AND ORGANIZATION

NOTE. It should be noted that in this paper the emphasis is on the boldface components of Figure 3: in particular, *no* emphasis will be placed on the direct algebraic/numeric solver that is used to solve the small subsystems specified by the DR-plan. Furthermore, in most of our discussion in both Parts I and II, we will restrict ourselves to variational spatial constraints. That is, we will *not* explicitly deal with history-based constraint mechanisms, although several of the desirable characteristics in $\mathcal{C}$, apply indirectly or directly (e.g. (ix)) to history-based constraint mechanisms as well.

In Sections 2 and 3, we precisely formulate the decomposition–recombination (DR) problem and lay down formal performance measures that reflect several of the nine characteristics $\mathcal{C}$ of DR-planners given above. These are not restricted to standard complexity measures. As noted before, to our knowledge, despite its longstanding presence, and the existence of prior DR-planners based on various *ad hoc* methods, the general DR-planning problem has not yet been clearly isolated or precisely formulated.

In Section 2, we formally define DR-*solvers* and their performance measures in the general context of polynomial systems arising from geometric constraints.

In Section 3, we give a parallel set of analogous definitions of DR-*planners* and their performance measures in the context of constraint graphs that incorporate geometric degrees of freedom.

In Section 4, we use the new performance measures to formally analyze two primary types of existing decomposition methods that are based on constraint graphs and degree of freedom analysis: constraint shape recognition (SR) and generalized maximum matching (MM). This analysis additionally provides intuitive substance to the newly defined performance measures.

NOTE. In Part II of this paper we describe and formally analyze three new DR-planners that were designed based on the performance measures described here. The development culminates with a new DR-planner called the modified frontier algorithm which excels with respect to these new performance measures.

2. Formal Definition of DR-solvers Using Polynomial Systems

This section will develop adequate notation for formally defining all of the terms that appear in italics and have been used informally so far. We will formally state the decomposition–recombination (DR) problem for polynomial systems arising from geometric constraints. This requires formalizing the notion of a decomposition–recombination solution sequence, and of decomposition–recombination solvers that fit the description $\mathcal{S}$ given in the previous section. In addition, we define the performance measures—that capture some of the desirable properties $\mathcal{C}$ given in the last section—for comparing such sequences and solvers.

NOTE. As mentioned in the organizational section, for the sake of gradual exposition, we will first assume that the constraint system is being solved even as the decomposition

is being generated—i.e. the DR-solver exactly fits the structural description $\mathcal{S}$ given in the previous section. Furthermore, we will define the DR-solver in the general context of polynomial equations that arise from the geometric constraints.

In Section 3 we will shift our attention to the DR-planning problem. The DR-planner generates a decomposition plan *a priori*, which then drives the direct algebraic/numeric solver—together, they form a DR-solver. The DR-planner, however, will be defined in the context of constraint graphs that incorporate geometric degrees of freedom. The DR-planner and its performance measures will be analogous to the *italics* terms defined here.

In order to formally define DR-solvers and their performance measures, we need to specify the model of real-algebraic computation used by the algebraic/numeric solver in Figure 3. However, the issue of the model or how real numbers are represented is entirely outside the focus of this manuscript for the following reason: our DR-solvers and performance measures are robust in that they are conceptually independent of the algebraic/numeric solver being used or how real numbers are represented. The definition of DR-solvers and performance measures adapts straightforwardly to other natural models of computation. In other words, our conceptual definition of DR-solvers and performance measures are such that if DR-solver A performs better than DR-solver B with respect to one of our performance measures, then it continues to do so irrespective of the (natural) model of computation being used by the algebraic/numeric solver.

Further, as pointed out in an earlier note, this paper only emphasizes the boldface components of Figure 3, specifically the DR-planner, whose operation defined in Section 3 will be seen to be purely combinatorial. In particular, no emphasis is placed on the non-boldface components, which includes the algebraic/numeric solver, and the model of computation it uses.

Having noted the above, the model of computation we assume—for the the sake of completeness and formality of definitions in this manuscript—is the Blum–Shub–Smale model of real algebraic computation (Blum *et al.*, 1989). Briefly, in this model, real numbers are assumed to be representable as such as entities, without any recourse to rational approximations and finite precision or interval arithmetic. Real arithmetic operations such as multiplication and addition and division can be performed in constant time, and finding an unambiguous representation of each real zero of a univariate polynomial p can also be achieved in time polynomial in the degree of p. Furthermore, these unambiguous representations of real zeroes of univariate polynomials are treated as entities just like any other real number, and can for instance, be used as coefficients of other polynomials.

A *system of equations* E is a pair (P, X) where P is a set of polynomial equations with real coefficients, and X is a set of formal indeterminates. The *union of two systems* $E_1 = (P_1, X_1)$ and $E_2 = (P_2, X_2)$ is the system $(P_1 \cup P_2, X_1 \cup X_2)$. An *intersection* $E_1 \cap E_2$ is the system $(P_1 \cap P_2, X_1 \cap X_2)$.

Within the geometric context, a solved subsystem is a rigid or solvable system where all the variables have already been "solved for", i.e. they have typically been expressed *explicitly* as polynomials of D free variables, where D represents the number of degrees of freedom of the rigid body (within its *local* coordinate system) in the prevailing geometry. In some cases, for example when the rigid body is a line in three dimensions, the number of free variables, including redundant ones, may be greater than D.

Let $E = (P, V)$ with $V = \{y_1, \ldots, y_D\} \cup X$ be such that every equation in P is of type $x_j = F_j(y_1, \ldots, y_D)$, for all $x_j \in X$ and F_j is a rational function (of the form

Q_1/Q_2 where Q_1 and Q_2 are, as usual, polynomials with real coefficients) that can be evaluated in time polynomial in $(|V|, |P|)$. Such a system of equations, E, is called a *solved system*. The variables $y_1, \ldots, y_D \in V$ of the solved system are *free variables* of E, and the variables $x_1, \ldots, x_k \in V$ are *explicitly fixed variables*. Note that all the variables in a solved system are fixed or free, whereas a general solvable system may have free, explicitly fixed and other, *implicitly fixed* variables.

A typical DR-solver—that follows the overall structural description $\mathcal{S}$ of Section 1.2—obtains a sequence $E_1, \ldots, E_m$, consisting of successively simpler solvable systems. These are general solvable systems and have successively fewer implicitly fixed variables; $E = E_1$; and E_m is a solved system. New variables y_i may be introduced at intermediate stages which represent free variables *within* subsystems S_i that are solved with respect to these variables. These solved subsystems represent various rigid bodies located and fixed with respect to their local coordinate systems. However, these local coordinate systems are still constrained with respect to each other, and hence in fact only D of the newly introduced variables are, in effect, free and the remainder are implicitly fixed. Some of these newly introduced variables may be removed at later stages. Eventually, in the solved system E_m, all the original variables and those newly introduced variables that might remain become explicitly fixed with respect to D free variables as well. The set of real solutions to E_{i+1} should also be a subset of solutions to E_i, to ensure that the final solutions to E_m actually represent solutions to the original system E.

We formally define real algebraic equivalence and real algebraic inference of two geometric constraint systems as follows. Given two systems $E_1 = (P_1, X \cup Y_1)$ and $E_2 = (P_2, X \cup Y_2)$—where the set X represents the original variables that are currently implicitly or explicitly fixed, and the sets Y_i represent the newly introduced variables (that are "free" *within* the solved subsystems S_i), we say that the system $E_1 = (P_1, X \cup Y_1)$ is *real algebraically inferable* (in short, inferable) from the system $E_2 = (P_2, X \cup Y_2)$, if for any real solution $Y_2 = A_2; X = B$ that satisfies the equations in P_2, there is a corresponding assignment A_1 of real values to Y_1 such that $Y_1 = A_1; X = B$ satisfies P_1. Two systems E_1 and E_2 are *real algebraically equivalent* if E_1 is real algebraically inferable from E_2 and E_2 is real algebraically inferable from E_1.

Now we are ready to define the notion of a DR-solution sequence. Let E be a system of equations.

A *DR-solution sequence* of E is a sequence of systems of equations $E_1, \ldots, E_m$ such that $E = E_1$, E_m is a solved system (so E_m has a real solution), every E_i is solvable, each E_i is inferable from E_{i+1}. Any solvable system E which has a real solution in fact has a DR-solution sequence. A trivial DR-solution sequence E, E^*—where E^* is a solved system equivalent to E—will do. (Note that by E_i we denote abstract algebraic systems, rather than their computer representations that could only have rational coefficients and therefore may only have approximate DR-solution sequences.)

The *DR-problem* is the problem of finding a DR-solution sequence of a given constraint system. A *DR-solver* is a constraint solver or algorithm that solves the DR-planning problem. A DR-solver is *general* if it always outputs a DR-solution sequence when given a solvable system as input. Aside from being general, we would also like a DR-solver to have the *Church–Rosser property*, i.e. the DR-solver should terminate irrespective of the order in which the solvable subsystems S_i are chosen. In other words, at each step, a solvable subsystem S_i can be chosen greedily to satisfy the (easily checkable) requirements of the algorithm. This prevents exhaustive search.

Next, we formally define a set of *performance measures* for the DR-solution sequences and DR-solvers. These performance measures are designed to capture the characteristics $\mathcal{C}$ of constraint solvers given in Section 1.3, that are desirable for engineering design and assembly applications. (Note that many of these measures are Boolean, i.e. either a certain desirable condition is met or not.) In particular, we would like a DR-solution sequence $Q = E_1, \ldots, E_m$ of a solvable system E to have several properties. To describe these properties we formalize a simplifying map from each system E_i to its successor E_{i+1}. In fact, for generality, we choose these maps T_i—called *subsystem simplifiers*—to map the set of subsystems of E_i onto the set of subsystems of E_{i+1}.

First, in order to reflect the Church–Rosser property in (i), and points (iv) and (vi) of $\mathcal{C}$, we would like these subsystem simplifiers T_i to be natural and well-behaved, i.e. to obey the following simple and non-restrictive rules.

(1) If A is a subsystem of B, then $T_i(A)$ is a subsystem of $T_i(B)$.
(2) $T_i(A) \cup T_i(B) = T_i(A \cup B)$.
(3) $T_i(A) \cap T_i(B) = T_i(A \cap B)$.

Second, in the description of the typical DR-solver $\mathcal{S}$ given in Section 1.2, each system E_{i+1} in the DR-solution sequence is typically to be obtained from E_i by replacing a solvable subsystem S_i in E_i (located during Step 1 of $\mathcal{S}$), by a simpler subsystem (during Steps 2 and 3). For a manipulable DR-solution sequence (again satisfying points (i), (iv), (v) and (vi) of $\mathcal{C}$), we would like E_{i+1} to look exactly like E_i outside of S_i. This leads to another set of properties desirable for the subsystem simplifiers T_i.

(4) Each $E_i = S_i \cup R_i \cup U_i$, $1 \leq i \leq m$, where S_i is solvable, R_i is a maximal subsystem such that S_i and R_i do not share any variables, S_i, R_i, U_i do not share any equations, and all variables of U_i are either variables of S_i or R_i. For any $A \subseteq R_i$, $T_i(A) = A$.

Thirdly, in order to address points (iv)–(vi), and (i) of $\mathcal{C}$ simultaneously, i.e. permitting generality of the subsystem S_i that is replaced by a simpler system during the ith step, while at the same time making it convenient for the designer to geometrically follow and manipulate the decomposition, we would like the subsystem simplifiers to satisfy the following property.

(5) For each i, all the pre-images of S_i, $T_j^{-1} \ldots T_{i-1}^{-1}(S_i), 1 \leq j \leq i-1$, are algebraically inferable from S_i, and furthermore, they are solvable or rigid subsystems for the given geometry (recall that the definition of "solvable" depends on the geometry). It follows from (2) and (3) above that the *inverse* $T_i^{-1}(A) = \bigcup B$ where the union is taken over all $B \subseteq E_i$, such that $T_i(B) \subseteq A$.

The above property states that while the subsystem simplifiers enjoy a high degree of generality and are completely free to map solvable subsystems into solvable systems, they should never map (convert) subsystems that are originally *not* solvable into one of the chosen, solvable systems S_i, at any stage i. In other words, in the act of simplifying, the subsystem simplifiers should not *create* one of the solvable subsystems S_i out of subsystems that were originally not solvable. A DR-solution sequence that satisfies the above properties is called *valid*. Thus a valid DR-solution sequence for a geometric constraint system E is specified as a sequence of $E_1, \ldots, E_m$ such that $E_1 = E$, E_m is a

solved system, every E_i is solvable and inferable from E_{i+1}, every $E_i = S_i \cup R_i \cup U_i$ as described above.

The motivation given for each of the properties above makes it clear that valid DR-solution sequences encompass highly general but geometrically meaningful solutions of the original constraint system E. Next we turn to point (ii) of $\mathcal{C}$, i.e. optimality, which also competes with generality (point (i)). For optimality, we would like to minimize the size of the largest solvable subsystem S_i in the DR-solution sequence. Formally, the *size* of the DR-solution sequence Q is equal to $\max_{1 \le i \le m} |S_i|$, where the size $|S_i|$ is equal to the total number of its variables less the number of its explicitly fixed variables. The *optimal DR-size* of the algebraic system E is the minimum size of Q, where the minimum is taken over all possible DR-solution sequences Q of E. An *optimal DR-solution sequence* Q of E is a solution sequence such that the size of Q is equal to the optimal DR-size of E. The *approximation factor* of the DR-solution sequence Q of the system E is defined as the ratio of the optimal DR-size of E to the size of Q.

As a general rule for optimality, it is clear that the larger the choice for solvable subsystems S_i of E_i available at any stage i, the more likely that one can find a *small* solvable S_i in E_i. In other words, we would like to make sure that the subsystem simplifier does *not destroy* solvability of too many *sub*systems starting from the original system E. Note that while the definition of DR-solution sequence makes sure that the entire system E_m is solvable if E_1 is, it does not require the same for subsystems of E_i. (In fact, even if all the E_i in a DR-solution sequence are algebraically equivalent, this would still not imply that solvability is preserved for the subsystems.) In addition, while the definition of a DR-solution sequence ensures that E_{i+1} has a real solution if E_i has one, it does not ensure the same for subsystems of E_i. The next two definitions capture properties of subsystem simplifiers that preserve subsystem solvability (resp. solutions) to varying degrees, thereby helping the optimality of the DR-solver, i.e. (ii) of $\mathcal{C}$. The DR-solution sequence $Q = E_1, \ldots, E_m$ is *solvability preserving* if and only if for all $A \subset E_i$, A is solvable (resp. has a real solution) and ($A \cap S_i = \emptyset$ or $A \subset S_i$) $\iff T_i(A)$ is solvable (resp. has a real solution). A DR solution sequence $Q = E_1, \ldots, E_m$ is *strictly solvability preserving* if and only if for all $A \subset E_i$, A is solvable (resp. has a real solution) $\iff T_i(A)$ is solvable (resp. has a real solution). Requiring such a solvability preserving simplifier places a weak restriction on the class of valid DR-solution sequences, but on the other hand, this restriction cannot eliminate all optimal DR-solution sequences. Furthermore, solvability preservation helps to ensure the Church–Rosser property in (i) of $\mathcal{C}$.

NOTE. To be more precise, "solvability preservation" should be termed "solvability and solution-existence preservation", but we choose the shorter phrase.

In fact, for DR-solution sequences to be optimal, we would prefer that for all $i \ge 1$, S_i does not contain any strictly smaller subsystem, say B that is solvable, and has not been found (during Step 1 of the description $\mathcal{S}$ in Section 1.2) and simplified/replaced at an earlier stage $j < i$. The DR-solution sequence $Q = E_1, \ldots, E_m$ is *complete* if and only if for every non-trivial solvable $B \subset S_i$, $B = T_{i-1} T_{i-2} \ldots T_j(S_j)$ for some $j \le i-1$. While the completeness requirement restricts the class of valid DR-solution sequences, it *only eliminates* sequences that either have size greater than optimal or the same size as some optimal sequence that does simplify B. In addition to affecting optimality, i.e. (ii) of $\mathcal{C}$, completeness also strongly reflects (ix): completeness prevents a DR-solver from

overlooking an overconstrained subsystem inside a well-constrained subsystem, which is also useful for constraint reconciliation (see Hoffmann and Joan-Arinyo, 1998).

So far, we have discussed performance measures that measure the desirability of DR-solution solution sequences. Next we formally define directly analogous performance measures for DR-solvers which generate DR-solution sequences. A DR-solver A is said to be *valid, solvability preserving, strictly solvability preserving, complete* if and only if for every input constraint system E, every DR-solution sequence produced by A is valid, solvability preserving, strictly solvability preserving or complete, respectively.

NOTE. Purely as a tool to help analysis and exposition, often we assume that DR-solvers for producing DR-solution sequences are randomized or non-deterministic in a natural way, i.e. those steps in the algorithm where *arbitrary* choices of equations or variables are made (for example, to be the lowest numbered equation or variable) are now taken to be randomized or non-deterministic choices.

The next definition formalizes performance measures related to characteristic (ii) of $\mathcal{C}$ that differentiates randomized DR-solvers for which *some* random choice leads to an optimal DR-solution sequence, from inferior DR-solvers where *no* choice would lead to an optimal DR-solution sequence. The *worst-choice approximation factor* of a DR-solver A on input system E is the minimum of the approximation factors of all DR-solution sequences Q of E obtained by the algorithm A over all possible random choices. The *best-choice approximation factor* of the algorithm A on input E is the maximum of the approximation factors of all the DR-solution sequences Q of E obtained by the algorithm A over all possible random choices.

3. Formal Definition of a DR-planner Using Constraint Graphs

The DR-solution sequence defined in the previous section embeds a DR-plan that is intertwined with the actual solution of the system. Therefore, a DR-solver that simply outputs a DR-solution sequence (even one that is strictly solvability preserving, complete, etc.), may not be modular as shown in Figure 3.

How does one construct a DR-solver that first generates a DR-plan before using it to drive the general algebraic/numeric solver? First note that repeatedly applying Steps 1 and 3 of the DR-solver in the description $\mathcal{S}$ in Section 1.2 could potentially generate a DR-plan without actually solving any subsystem (and without applying Step 2), provided that the following can be done:

(a) solvability of a subsystem S_i of the system E_i in Step 1 can be determined generically, without actually solving it; and
(b) a solvable subsystem S_i can be replaced in Step 3 without actually solving S_i—i.e. by a hypothetical solved subsystem—to give the new system E_{i+1}. For this we need a consistent and adequate *abstraction* of the systems E_i, and of their subsystems, as well as a way to abstract a hypothetical solved subsystem. In other words, we need to adapt the subsystem simplifier maps described in the last section, so that the iteration can proceed with Step 1 again.

By thus modifying the description $\mathcal{S}$ of the DR-solver, we obtain a DR-planner which can be employed as a pre-processing step to output a DR-plan instead of a DR-solution

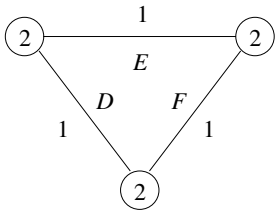

Figure 4. A constraint graph.

sequence. This DR-plan can *thereafter* be used to direct a series of applications of Step 2, leading to a complete DR-solution sequence. This modularity helps, for example, towards manoeuverability by the designer (point (vi)), and towards compatibility of the DR-solver with existing solvers (point (v)).

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.) Hence using constraint graphs facilitates *both* tasks (a) and (b) described above.

In other words a DR-planner that is based on a generalized degree of freedom analysis is robust in the following sense: changing the numerical values of constraints is not likely to affect the DR-plan. Thus the motivation for using constraint graphs includes all of the desirable characteristics (iii) to (vi) of the set $\mathcal{C}$ in Section 1.3. In addition, geometry is more visibly displayed via constraint graphs than via equations, thereby helping interaction with the designer.

NOTE. DR-plans and planners and their performance measures could also be defined directly in terms of the algebraic constraint systems, just as we defined DR-solvers. In this paper, however, due to the reasons mentioned above, we define DR-plans and planners entirely within the context of constraint graphs and degree of freedom analysis.

3.1. CONSTRAINT GRAPHS AND SOLVABILITY

First we define the abstraction that converts a geometric constraint system into a weighted graph. Recall that a geometric constraint problem consists of a set of geometric objects and a set of constraints between them.

A geometric constraint graph $G = (V, E, w)$ corresponding to a 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 weight of edge e, corresponding to the number of degrees of freedom available to an object represented by v and the number of degrees of freedom removed by a constraint represented by e, respectively. For example, Figure 4 is a constraint graph of a constraint problem shown in Figure 1. Note that the constraint graph could be a *hypergraph*, each hyperedge involving any number of vertices.

Now we introduce definitions that will help us to relate the notion of solvability of the geometric constraint system to the corresponding geometric constraint graph.

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. The function $d(A) = \sum_{e \in A} w(e) - \sum_{v \in A} w(v)$ is called the *density* of a graph A.

The constant D is typically $\binom{d+1}{2}$ where d is the dimension. The constant D captures the degrees of freedom associated with the *cluster* of geometric objects corresponding to the dense graph. In general, we use words "subgraph" and "cluster" interchangeably. 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 $D = 0$.

A dense graph with density strictly greater than $-D$ is called *overconstrained*. A graph that is dense and all of whose subgraphs (including itself) have density at most $-D$ is called *well-constrained*. A graph G is called *well-overconstrained* if it satisfies the following: G is dense, G has at least one overconstrained subgraph, and has the property that on replacing all overconstrained subgraphs by well-constrained subgraphs, G remains dense. A graph that is well-constrained or well-overconstrained is said to be *solvable*. A dense graph is *minimal* if it has no proper dense subgraph. Note that all minimal dense subgraphs are solvable, but the converse is not the case. A graph that is not solvable is said to be *underconstrained*. If a dense graph is not minimal, it could in fact be an underconstrained graph: the density of the graph could be the result of embedding a subgraph of density greater than $-D$.

In order to understand how solvable constraint graphs relate to solvable constraint systems it is important to remember that a geometric constraint problem has two aspects—combinatorial and geometric. The geometric aspect deals with actual parameters of the geometric constraint problem, while the combinatorial aspect deals with only the abstractions of objects and constraints. Unfortunately, at the moment it is not known how to completely separate the two aspects, except for some special cases. In this paper we will limit ourselves to the geometric constraint problems where generally there is a correspondence between solvable constraint systems and solvable constraint graphs. In order to do that, we need to introduce and formalize the notion of generic solvability of constraint systems. Informally, a constraint system is generically (un)solvable if it is (un)solvable for most choices of coefficients of the system. More formally we use the notion of *genericity* of e.g. Cox *et al.* (1998). A property is said to hold *generically* for polynomials $f_1, \ldots, f_n$ if there is a non-zero polynomial P in the coefficients of the f_i such that this property holds for all $f_1, \ldots, f_n$ for which P does not vanish.

Thus the constraint system E is generically (un)solvable if there is a non-zero polynomial P in the parameters of the constraint system—such that E is (un)solvable when P does not vanish. Consider for example Figure 5. Here the objects are six points in 2D and the constraints are eight distances between them. This system is unsolvable since the edge BC can be continuously displaced, unless the length of AD (induced by the lengths of AE, DE, DF, AF and EF) is equal to $|AB| + |BC| + |CD|$. Since we can create an appropriate non-zero polynomial $P(AB, BC, \ldots, EF)$, such that $P() = 0$ if and only if $|AD| = |AB| + |BC| + |CD|$, this system is generically unsolvable.

While a generically solvable system always gives a solvable constraint graph, the converse is not always the case. In fact, there are solvable, even minimal dense graphs whose corresponding systems are not generically solvable, and are in fact generically not solv-

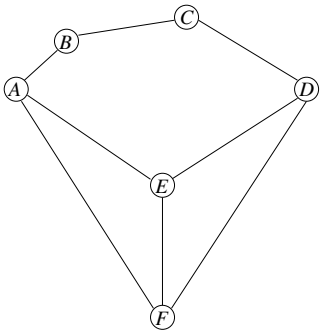

Figure 5. Generically unsolvable system.

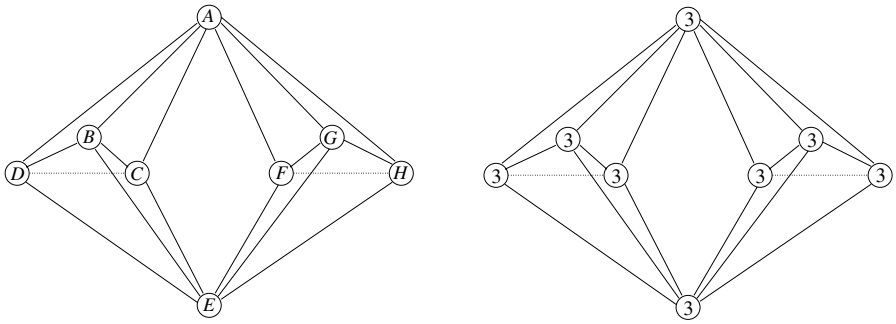

Figure 6. Generically unsolvable system that has a solvable constraint graph.

able (note that the position of the "not" changes the meaning, the latter being stronger than the former). Consider for example Figure 6. The constraint system is shown on the left, it consists of eight points in 3D and 18 distances between them. The corresponding constraint graph is shown on the right, the weight of all the edges is 1, the weight of the vertices is 3, the geometry-dependent constant $D = 6$. Note that this system is generically unsolvable since rigid bodies $ABCDE$ and $AFGHE$ can rotate about the axis passing through AE. This example can also be transformed into a constraint problem in 2D, where the objects are circles and the constraints are angles of intersections (Saliola and Whiteley, 1999).

Another example is the graph $K_{7,6}$ that in four dimensions represents distances between pairs of points. The constraint graph is minimal dense but it does not represent a generically solvable system.

It should be noted that in two dimensions, according to Laman's (1970) theorem, if all geometric objects are points and all constraints are distance constraints between these points, then any minimal dense subgraph represents a generically solvable system. In addition, there exists a purely combinatorial characterization of solvable systems in one dimension based on connectivity of the constraint graphs. However, as examples above indicate, the generalization of Laman's theorem fails in higher dimensions even for the case of points and distances.

There is a matroid-based approach to verifying whether solvability of the constraint graph implies solvability of the constraint system. It begins by checking whether a sub-

modular function $f(E) = 2|V|-3$, defined on sets of edges of the constraint graph, creates a matroid (Whiteley, 1992, 1997), i.e. checking whether the created function is positive on single edges. Thereafter this approach checks whether the corresponding geometric structure is generically rigid. Some matroid-based approaches to determining generic solvability are fast in practice, for example those to be discussed later in Section 4.2 under "Generalized Maximum Matching". However, so far no match has been established (for all cases) between the created matroid and generic rigidity of the particular geometric problem.

There are several attempts at characterization of generic solvability in dimension three and higher for the case of points and distances (Graver *et al.*, 1993; Tay, 1999). One such characterization, the so-called Henneberg construction, checks whether a given constraint graph can be constructed from the initial basic graph by applying a sequence of standard replacements. A characterization due to Dress checks whether the constraint graph satisfies a certain inclusion–exclusion type rule. A characterization due to Crapo examines whether the constraint graph can be decomposed into a union of certain edge-disjoint trees. All of these characterizations though interesting and useful, are so far unproven conjectures.

NOTE. Due to the above discussion, we restrict ourselves to the class of constraint systems where solvability of the constraint graph in fact implies the generic solvability of the constraint system. (As pointed out earlier, the converse is always true, with no assumptions on the constraint system.)

As was indicated above, this class is far from empty, it contains all constraint problems involving points and distances in 2D, problems resulting from Cauchy triangulations of the polyhedra in 3D as well as body-and-hinge structures in 3D. Moreover, it should be emphasized that while existing applications stop at finding subgraphs representing solvable constraint systems, we are interested in the entire problem of decomposition and recombination, optimizing the size of the largest subsystem to be solved, i.e. we are interested in finding an optimal DR-plan. In addition, note that for the class of generically solvable constraint systems and corresponding graphs, the DR-planning problem is already, in general, difficult (see a later note about NP-hardness, after the definition of the optimal DR-plan).

3.2. FORMAL DEFINITION OF DR-PLANS USING CONSTRAINT GRAPHS

Informally, stated in terms of constraint graphs, the DR-planning problem involves finding a sequence of graphs G_i—a DR-plan—such that the original constraint graph $G = G_1$ and every G_i contains a minimal solvable 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 be somehow equivalent to S_i, for example by having the same density value.) If the original graph G_1 is well-constrained, 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 minimal solvable subgraphs.)

Consider for example Figure 7 which shows one simplification step. On the left is the constraint graph G_1, the weight of all vertices is 2, the weight of all edges is 1, the geometry-dependent constant $D = 3$. Then $S_1 = \{A, B, C\}$ is a solvable subgraph. On

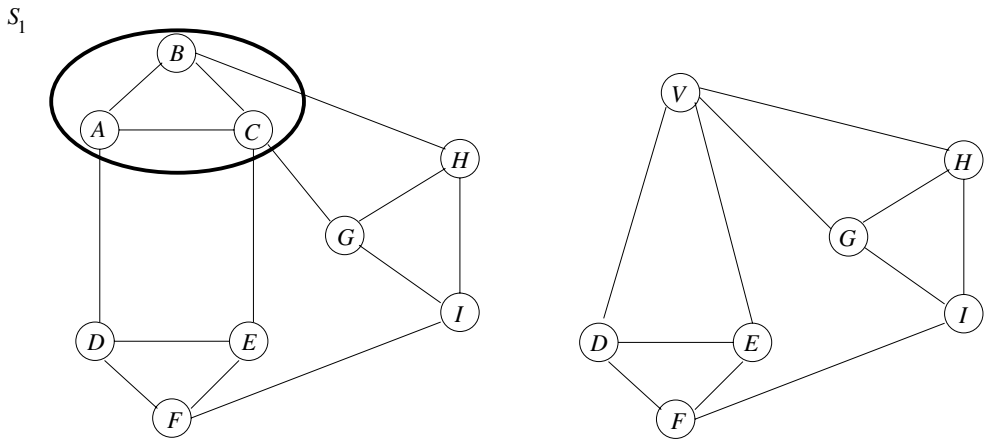

Figure 7. Original geometric constraint graph G_1 and simplified graph G_2.

the right is the simplified graph $T_1(G_1) = G_2$, after subgraph S_1 is replaced by a vertex $\{V\} = T_1(S_1)$. Since the density of the subgraph S_1 is -3, the weight of the vertex $\{V\}$ could be set to 3.

A sequence of simplification steps is shown in Figure 8. The top part depicts a geometric constraint graph G, where the weight of each edge is 1, the weight of each vertex is 2, and the dimension-dependent constant D is equal to 3 (this corresponds to points and distances in 2D). One of the plans for decomposing and recombining G (and the geometric constraint system that G represents) into small solvable subgraphs (representing solvable subsystems), is to decompose G into dense subgraphs $S_1 = \{A, B, C\}, S_2 = \{D, E, F\}, S_3 = \{G, H, I\}, S_5 = \{J, K, L\}, S_6 = \{M, N, O\}$; represent their solutions appropriately in a simplified graph so that they can be recombined, one possibility is to represent them as vertices P, Q, R, S, T of weight 3 each; recursively decompose the simplified graph into $S_4 = \{P, Q, R\}, S_7 = \{S, T\}$; represent and recombine their solutions as vertices U, W of weight 3; and so on until the entire graph is represented and recombined as a single vertex.

A corresponding DR-plan is shown at the bottom part of Figure 8. Note that there could be more than one DR-plan for a given constraint graph. For example, another possible DR-plan for a constraint graph described above is shown in Figure 9.

An optimal DR-plan will minimize the size of the largest solvable subgraph S_i found during the process. That is, it will *minimize the maximum fan-in of the vertices in the DR-tree* shown in Figures 8 and 9, where by fan-in of a vertex we mean the number of immediate descendants 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. Thus, such DR-plans satisfy the initial robustness requirements of characteristic (viii) of the set $\mathcal{C}$ described in Section 1.3.

Next we formally define a DR-plan for constraint graphs, and the various performance measures that capture desirable properties of DR-plans and DR-planners. The development is parallel to that in Section 2, where a DR-solution sequence and various perfor-

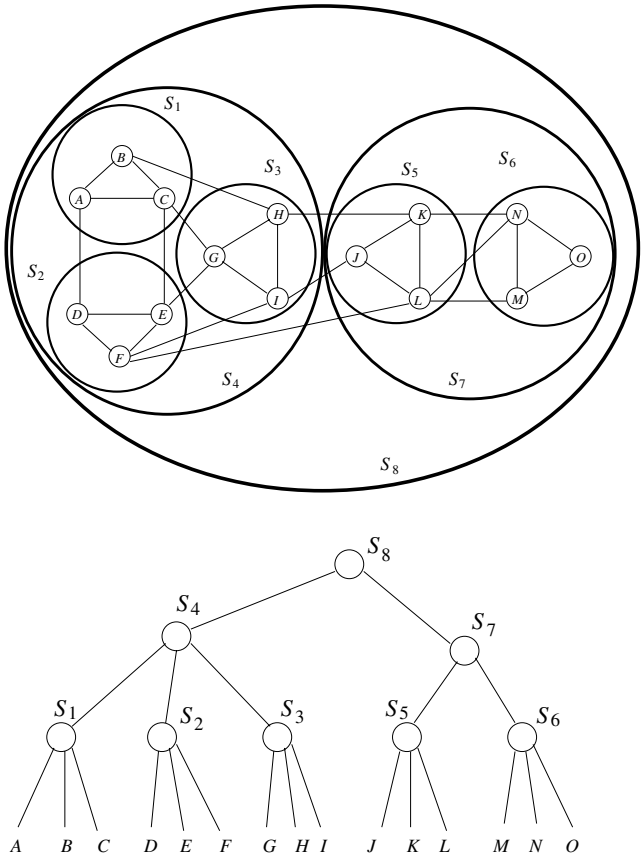

Figure 8. Geometric constraint graph and a DR-plan.

mance measures for these sequences and for DR-solvers were motivated and defined in terms of polynomial systems.

NOTE. Due to the strong analogy of DR-planners to DR-solvers defined in the previous section, the discussion here is more terse. The following are useful generic correspondences to keep in mind *after* reading the note at the end of Section 3.1, and the paragraphs preceding it:

Solvable subgraphs $\rightleftharpoons$ solvable subsystems;
DR-plans $\rightleftharpoons$ DR-solution sequences;
DR-planner $\rightleftharpoons$ DR-solver;
subgraph simplifier $\rightleftharpoons$ subsystem simplifier.

Let G be a solvable constraint graph. A *DR-plan* Q for G is a sequence Q of graphs $G_1, \ldots, G_m$ such that $G_1 = G$, G_m is a minimal solvable graph, and every G_i is solvable. An algorithm is a *general DR-planner* if it outputs a DR-plan when given a solvable constraint graph as input.

The *union (resp. intersection) of two subgraphs* A and B is the graph induced by the

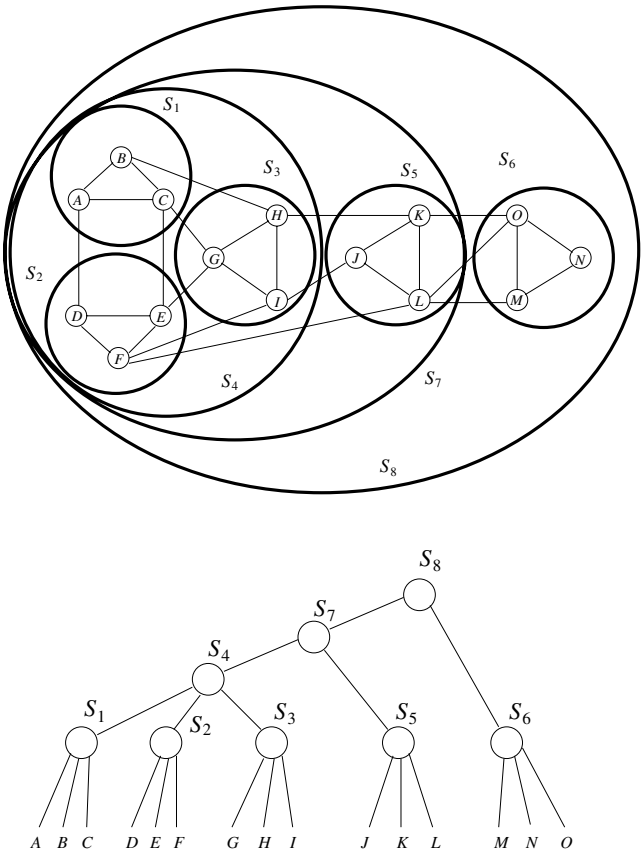

Figure 9. Another possible DR-plan.

union (resp. intersection) of sets of vertices of A and B. All subgraphs are understood to be vertex induced.

The mapping from the graph G_i to G_{i+1} is called a *subgraph simplifier* and is denoted by T_i. This mapping should have the following properties.

(1) If A is a subgraph of B, then $T_i(A)$ is a subgraph of $T_i(B)$.
(2) $T_i(A) \cup T_i(B)$ is the same as the graph $T_i(A \cup B)$.
(3) $T_i(A) \cap T_i(B)$ is the same as $T_i(A \cap B)$.

As in the case of DR-solution sequences, assume that every constraint graph G_i in the DR-plan can be written as $S_i \cup R_i \cup U_i$, where S_i is minimal solvable, R_i is a maximal subgraph such that S_i and R_i do not have common vertices, S_i, R_i, U_i do not have common edges and all vertices of U_i are either vertices of S_i or vertices of R_i. Analogous to properties (4) and (5) of the subsystem simplifier in the previous section, we would like the subgraph simplifier to have the following additional properties.

(4) For every $A \subseteq R_i$, $T_i(A) = A$.
(5) All the pre-images of S_i, i.e. $T_j^{-1} T_{j+1}^{-1} \dots T_{i-1}^{-1}(S_i)$ for all $1 \leq j < i-1$, are solvable.

A DR-plan that satisfies the above rules is called *valid*. The *size* of the DR-plan Q of G is the maximum of the sizes of S_i. The size of an arbitrary subgraph $A \subseteq G_i$ is computed as follows.

$Size(A) = 0$
For $1 \leq j \leq i-1$
  $B = T_{i-1}T_{i-2}\ldots T_j(S_j)$
  If $A \cap B \neq 0$
    then $Size(A) = Size(A) + D, A = A \setminus B$
  end if
end for
$Size(A) = Size(A) + \sum_{v \in A} w(v)$

In other words, the image of any of the S_j contributes D to the size of A, where D is a geometry-dependent constant, and the vertices of A that are not in any such image contribute their original weight. The *optimal size* of the constraint graph G is the minimum size of Q, where the minimum is taken over all possible DR-plans of G. An *optimal DR-plan* of G is the DR-plan that has size equal to the optimal size of G. The *approximation factor* of DR-plan Q of the graph G is defined as the ratio of the optimal size of G to the size of Q.

NOTE. The problem of finding the optimal DR-plan for a constraint graph with unbounded vertex weights is NP-hard. This follows from a result in the authors' paper (Hoffmann *et al.*, 1997) showing that the problem of finding a *minimum* dense subgraph is NP-hard, by reducing this problem to the CLIQUE. The CLIQUE problem is extremely hard to approximate (Hastad, 1996), i.e. finding a clique of size within a $n^{1-\epsilon}$ factor of the size of the maximum clique cannot be achieved in time polynomial in n, for any constant ϵ (unless $P = NP$). However our reduction of CLIQUE to the optimal DR-planning problem is not a so-called gap-preserving reduction (or L-reduction); thus how well this problem could be approximated is still an open question.

The definition of solvability preservation is largely analogous to the case of DR-solvers, but is additionally motivated by the following. In the case of DR-solvers, one condition on a solvability preserving simplifier is that it preserves the existence of real solutions for certain subsystems. Here, in the case of DR-plans, a natural choice is to correspondingly require that the simplifier does not map well-constrained subgraphs or overconstrained subgraphs to underconstrained and vice versa. The DR-plan Q of G is *solvability preserving* if and only if for all $A \subset G_i$, A is solvable and ($A \cap S_i = \emptyset$ or $A \subset S_i$) $\Longleftrightarrow$ $T_i(A)$ is solvable. The DR-plan Q of G is *strictly solvability preserving* if and only if for all $A \subset G_i$, A is solvable $\Leftrightarrow$ $T_i(A)$ is solvable. The DR-plan Q of G is *complete* if and only if for all non-trivial solvable $B \subset S_i$, $B = T_{i-1}T_{i-2}\ldots T_j(S_j)$ for some $j \leq i-1$.

Next, we formally define DR-planners and their performance measures. An algorithm is said to be a *DR-planner* if it outputs a DR-plan when given a solvable constraint graph as input. As before, we consider DR-planners to be randomized algorithms. A randomized DR-planner A is said to be *valid, solvability preserving, strictly solvability preserving, complete* if and only if for every G *every* DR-plan produced by A is valid, solvability preserving, strictly solvability preserving or complete, accordingly. The *worst-choice approximation factor* of a DR-planner A on input graph G is the minimum of

the approximation factors of all DR-plans Q of G obtained by the DR-planner A over all possible random choices. The *best-choice approximation factor* of the algorithm A on input graph G is the maximum of the approximation factors of all the DR-plans Q of G obtained by the DR-planner A over all possible random choices.

In addition to the above performance measures, we define three others that reflect the Church–Rosser property, the ability to deal with underconstrained subsystems, as well as the ability to incorporate an input, design decomposition provided by the designer.

A DR-planner is said to have the *Church–Rosser* property, if the DR-planner terminates with a DR-plan irrespective of the order in which the dense subgraphs S_i are chosen.

A DR-planner A *adapts to underconstrained constraint graphs* G if every (partial) DR-plan produced by A terminates with a set of solvable subgraphs Q_i such that each solvable subgraph Q_i has no supergraph that is solvable, and moreover, no subgraph of G that is disjoint from all of the Q_i's is solvable.

A *conceptual design decomposition* P is a set of solvable subgraphs P_i, which are partially ordered with respect to the subgraph relation. A DR-planner A is said to *incorporate a design decomposition* P, if for every DR-plan Q produced by A, the corresponding sequence of solvable subgraphs S_i embeds a topological ordering of P as a subsequence—recall that a topological ordering is one that is consistent with the natural partial order given by the subgraph relation on P.

When a DR-plan incorporates a design decomposition P, the level of a cluster P_i in the partial ordering of P can now be viewed as a priority rating which specifies which component of the design decomposition has most influence over a given geometric object. In other words, a given geometric object K is first fixed/manipulated with respect to the local coordinate system of the lowest level cluster $P_i \in P$ containing K. Thereafter, the entire cluster P_i can be treated as a unit and can be independently fixed/manipulated in the local coordinate system of the next level cluster P_j containing (the simplification of) P_i, etc.

Finally, we summarize how the above formal performance measures capture the informally described characteristics in the set $\mathcal{C}$ given in Section 1.3. The property of being a general DR-planner refers to whether the method successfully terminates with a DR-plan in the general case, and reflects characteristic (i); since we use constraint graphs which yield robust DR-plans that can be obtained efficiently, the property of being a general DR-planner also reflects (iii) and (viii); dealing with underconstrained graphs also reflects (i); incorporating input, design decompositions reflects (v); validity influences the Church–Rosser property and reflects (i) as well as (iv), (v), (vi); solvability preservation, strict solvability preservation influences the Church–Rosser property and reflects (i), (ii), (iv), (v); completeness is based on criteria (ii) and (ix); worst- and best-choice approximation factors are based on (ii) and complexity directly reflects (iii).

4. Performance Analysis of Prior DR-planners

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

NOTE. Due to reasons discussed in Section 3, we leave out those graph rigidity-based methods for distance constraints in dimensions three or greater such as Tay and Whiteley (1985) and Hendrickson (1992) as well as methods such as Crippen and Havel (1988), Havel (1991) and Hsu (1996) since they are non-deterministic or rely on symbolic compu-

tation, or they are randomized or generally exponential and based on exhaustive search. Graph rigidity- and matroid-based methods for more specific constraint graphs are discussed in Section 4.2 under the so-called maximum matching-based algorithms. Also, for reasons discussed in the Introduction, we leave out methods based on decomposing sparse polynomial systems.

The first type of algorithm, which we call SR for *constraint shape recognition* (e.g. Owen, 1991, 1996; Hoffmann and Vermeer, 1994, 1995; Bouma *et al.*, 1995; Fudos and Hoffmann, 1996b, 1997), concentrates on recognizing specific solvable subgraphs of known shape, most commonly, patterns such as triangles. The second type, which we call MM for *generalized maximum matching* (e.g. Kramer, 1992; Ait-Aoudia *et al.*, 1993; Pabon, 1993; Latham and Middleditch, 1996), is based on first isolating certain solvable subgraphs by transforming the constraint graph into a bipartite graph and finding a maximum generalized matching, followed by a connectivity analysis to obtain the DR-plan. In this section, we give a formal performance analysis of SR- and MM-based algorithms—choosing a representative algorithm (typically the best performer) in each class—using the performance measures defined in the previous section.

Informally, one major drawback of the SR and MM algorithms is their inability to perform a generalized degree of freedom analysis. For example, SR would require an infinite repertoire of patterns. In the case of spatial constraints, some elementary patterns have been identified (Hoffmann and Vermeer, 1994, 1995). In the case of extending the scope of planar constraint solvers, adding free-form curves or conic sections requires additional patterns as well (Hoffmann and Peters, 1995; Fudos and Hoffmann, 1996a). Similarly, a decomposition of underconstrained constraint graphs into well-constrained components is possible for SR algorithms but only subject to the pattern limitations. In many cases, MM algorithms will output DR-plans with larger, non-minimal subgraphs S_i that may contain smaller solvable subgraphs. This affects the approximation factors adversely.

This inability to find general minimal dense subgraphs also affects their ability to deal with overconstrained subgraphs that arise in assemblies, which is in turn needed to perform constraint reconciliation (Hoffmann and Joan-Arinyo, 1998).

4.1. constraint shape recognition (SR)

Consider the algorithm of Fudos and Hoffmann (1996b, 1997) which relies on the following strong assumption *(SR1)*: all geometric objects in two dimensions have 2 degrees of freedom and all constraints between them are binary and destroy exactly 1 degree of freedom. Thus, in the corresponding constraint graph, the weight of all the edges is 1 and of all the vertices is 2. Because of this assumption, the SR algorithm ignores the degrees of freedom and relies only on the topology of the constraint graph.

description of the algorithm

We give a terse description of the algorithm—the reader is referred to Fudos and Hoffmann (1996b, 1997) for details. Our description is meant only to put the algorithm into an appropriate DR-planner framework that is suited for the performance analysis.

The algorithm consists of two phases. During Phase One, SR uses the bottom-up iterative technique of Itai and Rodeh (1978): in the current graph G_i (where $G_1 = G$),

specific solvable graphs (clusters) are found that can be represented as a union of three previously found clusters that pairwise share a common vertex. Such configurations of three clusters are called *triangles*. The vertices of G_i represent clusters and edges of G_i represent constraints between clusters (initially due to SR1, every vertex and every edge of G is a cluster). More specifically, once a triangle formed by three clusters has been found, the three vertices in G_i corresponding to these three clusters are simplified into one new vertex in the simplified graph G_{i+1}. The edges of G_{i+1} are induced by the edges of the three old vertices. This is repeated for k steps until there are no more triangles left. If there is only one cluster left in G_k, then the algorithm terminates. Otherwise G_k serves as an input to Phase Two of the algorithm.

Before we describe Phase Two, we note that a so-called *cluster graph* C_i corresponding to G_i is used as an auxiliary structure for the purpose of finding triangles. Hence the partial DR-plan produced during Phase One of the algorithm is of the form: $(G_1, C_1), \ldots, (G_k, C_k)$. The vertices of the cluster graph C_i correspond to

- vertices of the original graph G_1;
- cluster vertices in the graph G_i; and
- edges in G_1 which have not been included into one of the cluster vertices in the graphs $G_{i-1}, \ldots, G_1$.

In particular, the first cluster graph C_1 contains one vertex for every vertex and every edge of G_1. The edges of the cluster graph C_i connect the vertices of G_i that represent clusters to the original vertices from G_1 that are contained in these clusters. Due to the structure of cluster graphs, triangles in G_i are found by looking for *specific* 6-cycles in the corresponding cluster graph C_i. These 6-cycles consist of three cluster vertices and three original vertices. The new cluster graph C_{i+1} is constructed from C_i by adding a new vertex c_i representing the newly found cluster S_i, and connecting it by edges to the original vertices from G_1 that are in the cluster S_i. We also remove the three old cluster vertices in C_i which together formed the new cluster S_i. We note that this is the *only* way in which cluster vertices are removed from cluster graphs. In particular, situations may arise where two clusters (that share a single vertex) are represented by the same vertex in the graph G_i, but they are represented by distinct vertices in the cluster graph C_i.

During Phase Two, SR constructs the remainder of the DR-plan $(G_k, C_k), \ldots, (G_m, C_m)$. First SR uses a global top–down technique of Hopcroft and Tarjan (1973) to divide G_k into a collection of triconnected subgraphs. These subgraphs are found by recursively splitting the original graph using separators of size at most 2. Thus the triconnected components can be viewed as the leaves of a binary tree T each of whose internal nodes corresponds to a vertex separator of size at most 2.

For every triconnected subgraph S in G_k, a new cluster vertex is created in the cluster graph C_k: similar to Phase One, this vertex replaces all vertices that represent existing clusters contained in S.

The next pair (G_{k+1}, C_{k+1}) in the DR-plan is found as in Phase One by finding a triangle in G_k, or effectively locating a 6-cycle in C_k. However, triangles in G_k have already been located in the course of constructing the tree T—these triangles were originally split by the vertex separators at the internal vertices. Thus, if the original constraint graph G is solvable, the remainder of the DR-plan $(G_{k+1}, C_{k+1}), \ldots, (G_m, C_m)$ is formed by repeated simplifications along a bottom up traversal of the tree T.

If the original graph was underconstrained, then it is still possible to construct a

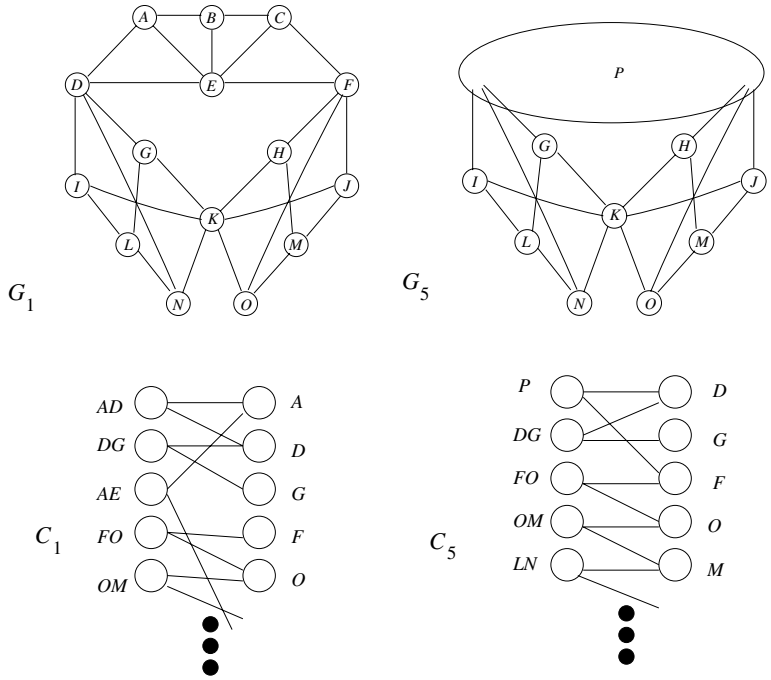

Figure 10. The original graph $G = G_1$, the cluster graph C_1 and the simplified graphs G_5, C_5.

DR-plan of its maximal well-constrained subgraphs, by introducing additional constraints and making the original graph solvable. That is, in order to complete the bottom up traversal of the tree T in Phase Two, additional constraints need to be introduced, to make the whole graph solvable. For details, see Fudos and Hoffmann (1996b, 1997).

EXAMPLE

Consider Figure 10. During Phase One, the triangles ADE, ABE, BCE and CEF will be discovered and simplified as P. During Phase Two, the remainder of the graph will be divided into triconnected subgraphs $P, PGKILN$ and $OKHMJP$, then $PGKILN$ and $OKHMJP$ are simplified and finally the union of $P, PGKILN$ and $OKHMJP$ is simplified.

DEFINING THE SIMPLIFIER MAP

The same simplifier is used throughout Phase One and Two: replace a subgraph S_i consisting of a triangle of clusters in G_i (or triconnected subgraphs during Phase Two) by one vertex in G_{i+1} representing this triangle. The subgraph S_i is found as a 6-cycle in the cluster graph C_i. The cluster graph C_{i+1} is constructed as described in Phase One.

More formally, recalling the definitions in Section 3: let G be a constraint graph; the first graph G_1 in the DR-plan is the original graph G. Let $G_i = (V, E)$ be the current graph and let S_i be a cluster found. Let $A \subseteq G_i, A = (V_A, E_A)$. Then the image of A under the subgraph simplifier T_i is $T_i(A) = A$, if the intersection of A and S_i is empty; otherwise $T_i(A) = (V_{T_i(A)}, E_{T_i(A)})$ where $V_{T_i(A)}$ is the set of all vertices of A that are

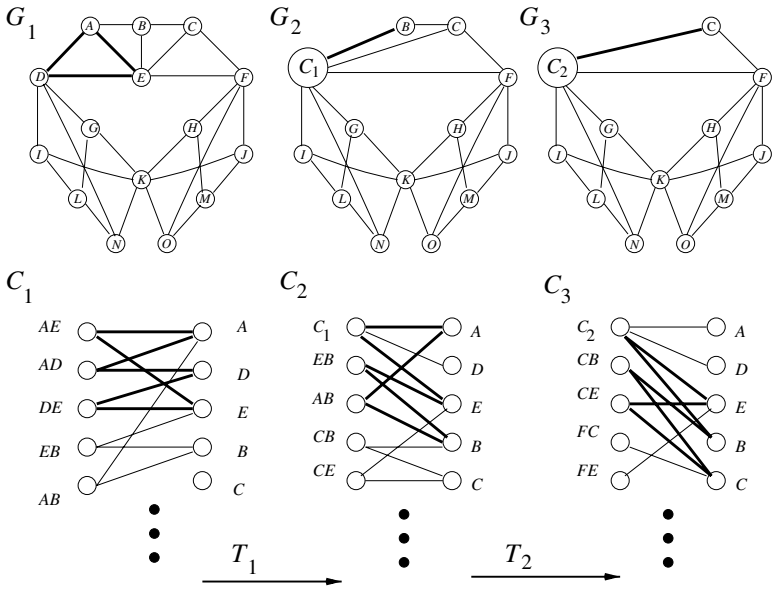

Figure 11. Action of the simplifier on G_i and C_i during Phase One.

not vertices of S_i plus a vertex c_i that replaces the cluster S_i. The set of edges $E_{T_i(A)}$ is a set of all edges of A that are not edges of S_i; the edges of E_A that have exactly one endpoint in S_i will have this endpoint replaced by the vertex c_i; and the edges of A that have both endpoints in S_i are removed.

Since the cluster S_i in G_i is located by finding a 6-cycle in C_i, we need to describe how C_{i+1} is constructed from C_i, i.e. the effect that T_i has on C_i (formally, T_i is a map from (G_i, C_i) to (G_{i+1}, C_{i+1})). To obtain C_{i+1}, we start with C_i and first add a new vertex c_i representing the cluster S_i, which is connected by edges to all the original vertices that it contains. Finally, vertices in C_i—that represent clusters entirely contained in S_i—are removed, and edges adjacent to these vertices are also removed.

Figures 11 and 12 illustrate the action of the subgraph simplifier T_i on both G_i and C_i.

A FURTHER CONCESSION FOR SR

Observe that the SR algorithm is not a general DR-planner when input geometric constraint graphs do not comply with assumption SR1. For example, for the graph shown in Figure 13, during Phase One, SR would not find any triangles and during Phase Two, it would conclude that the graph is underconstrained, even though the graph is well-constrained.

SR implementations may remedy some cases similar to this one. For instance, weight 2 edges to weight 2 vertices often arise from incidence constraints between geometric elements of the same type (two coincident points or two coincident lines), and such cases are easily accounted for by working with equivalence classes of such vertices. Moreover, most planners will evaluate the density of a graph before announcing under- or over-constrained situations. Thus, an implementation of the SR algorithms of Fudos and Hoffmann (1996b, 1997) may or may not construct a DR-plan for the graph of Figure 13

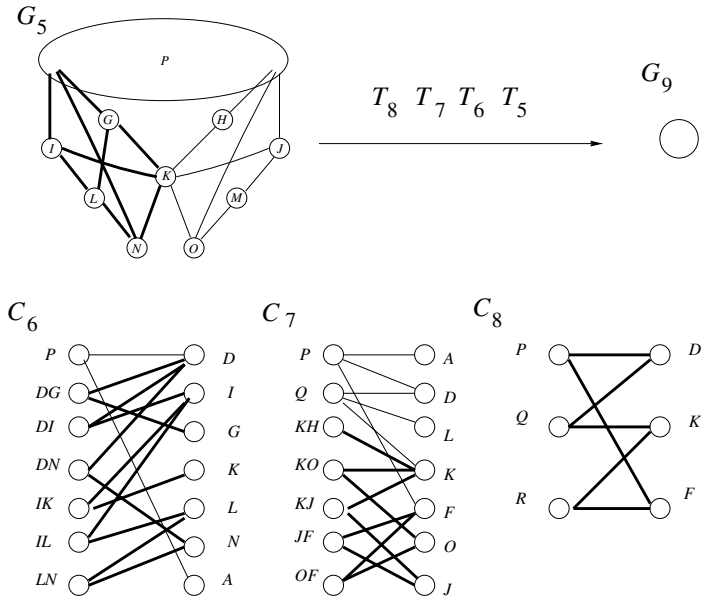

Figure 12. Action of the simplifier during Phase Two.

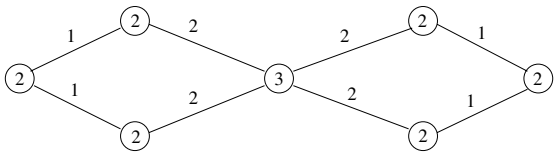

Figure 13. This solvable graph would not be recognized as solvable by SR.

depending on the original problem statement. It is clear that such heuristics enlarge the class of solvable graphs, but fall short of decomposing *all* solvable constraint graphs.

However, even when input graphs do comply with SR1, and SR checks the overall density of a graph, SR could *still* mistake a graph that is not solvable for solvable. Figure 14 shows an example which the SR algorithm may process incorrectly: since the graph contains no solvable triangles, SR proceeds immediately to Phase Two. During Phase Two, the triconnectivity of the entire graph could be erroneously construed to mean that the graph is solvable. In fact, the graph does have density -3, which superficially seems to support such a conclusion. However, the graph is certainly not well-constrained: the eight vertices on the right form a subgraph of density -2; that is, an overconstrained subgraph. Moreover, if this overconstrained subgraph is replaced a subgraph of density -3, the resulting graph has density -4 revealing that it is not well-overconstrained either, and therefore not solvable.

Figure 14 demonstrates that the density calculation heuristics is insufficient to determine the existence of minimal dense subgraphs of triconnected constraint graphs. What is needed is a general algorithm for finding minimal dense subgraphs.

In order to give a performance analysis of the SR algorithm for those classes of constraint graphs where it *does* produce a satisfactory DR-plan, we make a *strong concession (SR2)*

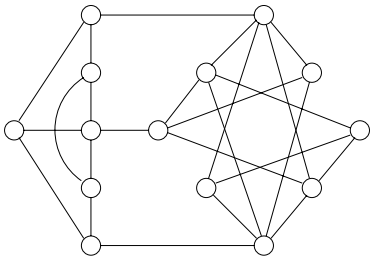

Figure 14. Weight of all vertices is 2, weight of all edges is 1.

that: only "triangular" structures are "acceptable" for the remainder of this section. That is, we modify the definitions of validity, solvability preserving, strictly solvability preserving and completeness by replacing the words "solvable subgraph" by the following recursive definition: "either a subgraph that can be simplified into a single vertex by a sequence of consecutive simplifications of triangles of solvable subgraphs (using the simplifier described in Section 4.1), or a vertex, edge or triconnected subgraph".

PERFORMANCE ANALYSIS

In this section, we analyze the SR algorithm with respect to the various performance measures defined in Section 3.

CLAIM 4.1. *Under the concession SR2, the SR algorithm is valid.*

PROOF. We will show that every pre-image—of the "solvable" cluster S_i found at the ith iteration of the SR algorithm—is also "solvable", using the stricter SR2 definition of "solvable". It can be easily checked that the other requirements necessary for validity from Section 3 also clearly hold.

Let S_i be a cluster that has been found in G_i by the SR algorithm (by locating a 6-cycle in C_i). Then from the description of the algorithm, and by assertion SR2, there is a sequence of simplifications, say $T_{i+1}, \ldots, T_m$ whose composition maps S_i to a single vertex. For every pre-image $A \subseteq G_j$ $A = T_j^{-1} \ldots T_{i-1}^{-1}(S_i)$, for $j \leq i-1$, the composition $T_m \circ \cdots \circ T_i \circ T_{i-1} \circ \cdots \circ T_j$ will map A to a single vertex, hence A is "solvable". □

CLAIM 4.2. *The SR algorithm is strictly solvability preserving under the concession SR2, and therefore is solvability preserving as well.*

PROOF. Let A be a "solvable" subgraph, i.e. there is a sequence of simplifications $A_1, \ldots, A_m$ such that $A = A_1$ and A_m consists of only one vertex. If the intersection of A and the currently found cluster S_i is empty, then $T_i(A) = A$ and it remains "solvable". If this intersection is not empty, then a subgraph $B = A \cap S_i$ is simplified into a new cluster vertex $c_i = T_i(S_i)$. We need to show that in this case $T_i(A)$ remains "solvable" as well. Suppose that some subgraph of B is a vertex of a 6-cycle in one of the cluster graphs formed during the simplification $A_1, \ldots, A_m$. Then clearly the new cluster vertex c_i could perform the same function: i.e. it could also be a vertex in that 6-cycle of one of the cluster graphs formed during the simplification $A_1, \ldots, A_m$. In addition, if A was triconnected, then so is $T_i(A)$. This proves that there is a sequence (essentially a

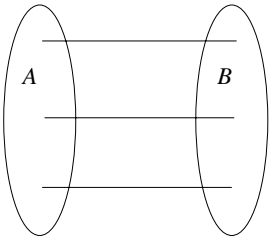

Figure 15. Solvable graph containing two triconnected subgraphs not composed of triangles.

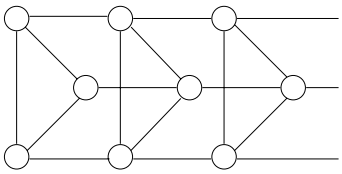

Figure 16. Solvable graph consisting of $n/3$ solvable triangles.

sequence $A_1, \ldots, A_m$ modified by replacing B by c_i) which terminates in a single vertex, thus demonstrating the "density" of $T_i(A)$. □

Claim 4.3. *Even under concession SR2, SR is not complete.*

Proof. We will show that there are cases when the SR picks large, non-minimal "solvable" subgraphs S_i to simplify, ignoring smaller "solvable" subgraphs of S_i.

Consider Figure 15. If subgraphs A and B are triconnected but not composed of triangles, then they are "solvable", but since the entire graph is triconnected neither A nor B will be simplified by SR. However, since the whole graph $A \cup B$ is triconnected, it will chosen by SR as S_1 during Phase Two. □

Claim 4.4. *The (worst- and) best-choice approximation factor of SR under concession SR2 is at most* $O(\frac{1}{n})$.

Proof. Consider Figure 16. The entire graph consists of $n/3$ triangles. During Phase One, SR will successfully locate and simplify each one of them. However, during Phase Two, SR will not be able to decompose the entire solvable graph into smaller solvable ones (since the entire graph is triconnected) so the size of the corresponding DR-plan is $O(n)$. On the other hand, the optimal DR-plan would simplify neighboring pairs of triangles, one pair at a time, thus the optimal size is a small constant. □

Next, we consider three other performance measures discussed in Section 3.

Claim 4.5. *Under the concession SR2, the algorithm SR adapts to underconstrained graphs.*

Proof. Suppose that the graph G is underconstrained. Let A be a "solvable" subgraph that is not contained in any other "solvable" graph. Since Phase Two of the SR algorithm

is top–down, either SR will find A and simplify it as one of the S_i's, or $A \cap S_i \neq \emptyset$ for one of the S_i. In either case, SR adapts to G. □

Observation 4.6. *Under the concession SR2, the SR algorithm has the Church–Rosser property, since the new graph G_{i+1} remains "solvable" if G_i is "solvable", regardless of the choice of the "solvable" S_i that is simplified at the ith stage.*

Claim 4.7. *Under the concession SR2, the SR algorithm can incorporate design decompositions P if and only if P fulfills the following requirement: any pair of "solvable" subgraphs P_k and P_t in P satisfy $P_k \subseteq P_t$ or $P_t \subseteq P_k$ or $P_k \cap P_t$ contains no edges.*

Proof. For the "if" part we consider the most natural modification of the original SR algorithm, and find a topological ordering O of the given design decomposition P—which is a set of "solvable" subgraphs of the input graph G, partially ordered under the subgraph relation—such that O is embedded as a subplan of the final DR-plan generated by this modified SR algorithm; i.e. O forms a subsequence of the sequence of "solvable" subgraphs S_i, whose (sequential) simplification gives the DR-plan.

We take any topological ordering O of the given design decomposition P and create a DR-plan for the first "solvable" subgraph P_1 in P. That is, while constructing the individual DR-plan for P_1, we "ignore" the rest of the graph. This individual DR-plan induces the first part of the DR-plan for the whole graph G. In particular, the last graph in this partial DR-plan is obtained by simplifying P_1 using the simplifier described in the section Defining the Simplifier Map (and treating P_1 exactly as SR would treat a cluster S_j found at some stage j). Let G_i be the last graph in the DR-plan for G created thus far. Now, we consider the next subgraph P_2 in the ordering O, and find an individual DR-plan for it, treating it not as a subgraph of the original graph G, but as subgraphs of the simplified graph G_i. This individual DR-plan is added on as the next part of the DR-plan of the whole graph G.

The crucial point is that the simplification of any subgraph, say P_k, will not affect any of the unrelated subgraphs $P_t, t \geq k$, unless $P_k \subseteq P_t$. This is because, by the requirement on P, P_k and P_t share no edges. Therefore, when the cluster vertex for P_k is created, none of the clusters inside P_t are removed.

The process—of constructing individual DR-plans for subgraphs in the decomposition P and concatenating them to the current partial DR-plan—is continued until a partial DR-plan for the input graph G has been produced, which completely includes topological ordering O of the decomposition P as a subplan. Let G_p be the last graph in this partial DR-plan. The rest of the DR-plan of G is found by running the original SR algorithm on G_p and the corresponding cluster graph C_p.

For the "only if" part, consider Figure 17. Let $P=\{P_0, P_1, P_2\}$, where $P_0 = ABD, P_1 = BCD, P_2 = ABCD$. Then SR cannot produce any DR-plan that would incorporate P as subplan. □

4.2. generalized maximum matching (MM)

Consider the algorithms of Ait-Aoudia *et al.* (1993), Pabon (1993), Kramer (1992), Serrano and Gossard (1986) and Serrano (1990) as well as graph rigidity- and matroid-based methods for distance constraints in two dimensions (Imai, 1985; Gabow and Westermann,

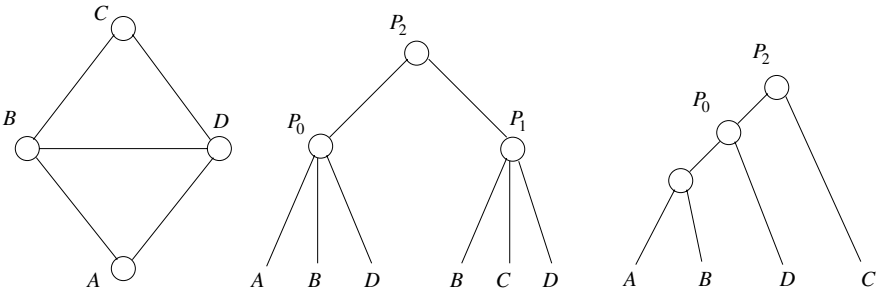

Figure 17. Constraint graph, intended and actual decompositions.

1988; Hendrickson, 1992) as well as more general constraints (Sugihara, 1985) all of which more or less use (generalized) maximum matching (or equivalent maximum network flow) for finding solvable subgraphs in specialized geometric constraint graphs. These methods either assume that the constraint graph has zero density or they reduce the weight of an arbitrarily selected set of vertices in order to turn the graph into one of zero density.

In this paper we will analyze what we consider to be the most general algorithm of this kind (Pabon, 1993) (it generalizes the algorithm of Ait-Aoudia *et al.*, 1993, although the latter provides a more complete analysis), supplemented (by us, as suggested by a reviewer) with a method from Hendrickson (1992).

Note that while the algorithm of Pabon (1993) both locates solvable subgraphs and describes how to construct a corresponding DR-plan, Sugihara (1985), Hendrickson (1992) and Gabow and Westermann (1988) only describe algorithms that allow one to verify whether a given graph is solvable, but do not explicitly state how to use these algorithms for successively decomposing into small solvable subgraphs, i.e. for constructing DR-plans.

While neither of the algorithms analyzed in this paper perform well according to our previously defined set of criteria, later, in Part II of this paper, we will describe our network flow-based algorithm MFA that improves the performance in several key areas.

DESCRIPTION OF THE ALGORITHM

As in the case of SR, we give a terse description of the MM algorithm—the reader is referred to Ait-Aoudia *et al.* (1993) and Pabon (1993) for details. Our description is meant only to put the algorithm into an appropriate DR-planner framework that is suited for the performance analysis.

By using maximum flow, the input constraint graph is decomposed into a collection of subgraphs that are strongly connected. The flow information also provides a partial ordering of the strongly connected subgraphs representing the order in which these subgraphs should be simplified. This ordering in turn specifies the DR-plan. It is important to note that these strongly connected components represent a sequence of solvable subgraphs of the original constraint graph, *only if* the input constraint graph is well-constrained.

Consider Figure 18. All the vertices have weight 2, all the edges have weight 1, and the geometry is assumed to be in two dimensions (i.e. the geometry-dependent constant $D = 3$). The output of the MM algorithm is:

- a set of vertices whose total weight is reduced by 3 units, say weight of vertices

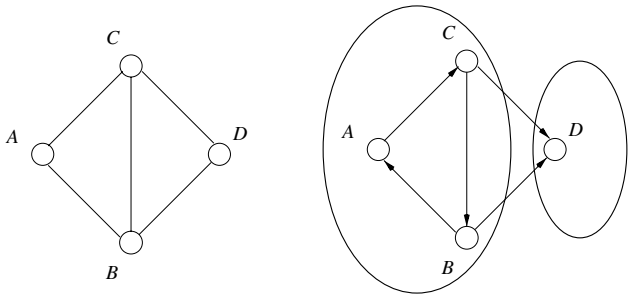

Figure 18. Original constraint graph and its decomposition into strongly connected components.

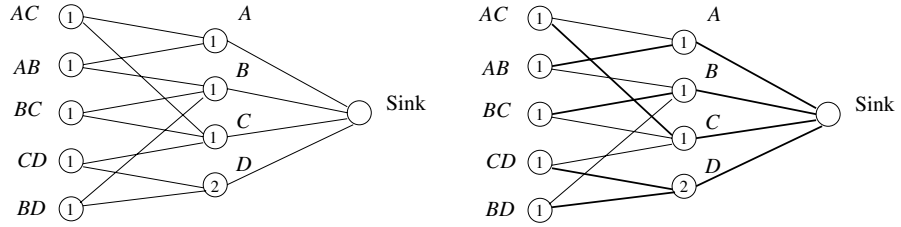

Figure 19. Modified bipartite graph and maximum flow in this graph.

A, B and C to be reduced by 1 unit each, (this corresponds to fixing 3 degrees of freedom—the number of degrees of freedom of a rigid body in two dimensions);
- two strongly connected components ABC and D;
- the DR-plan, i.e. the information that the subsystem represented by ABC should be simplified/solved first and then its union with D should be simplified/solved.

In order to produce such an output, the MM algorithm first constructs the network $X = (VX, EX)$ corresponding to the original weighted constraint graph $G = (V, E)$ (after the weights of A, B, C were reduced by 1 unit each). The set of vertices VX is the union of VX_1 and VX_2 where the vertices in VX_1 correspond to the vertices in V, vertices in VX_2 correspond to the edges in E. An edge $ex \in EX$ would be created between $vx_1 \in VX_1$ and $vx_2 \in VX_2$ if the vertex in V corresponding to the vx_1 is an endpoint of an edge in E corresponding to vx_2. The edge ex has infinite capacity. All the vertices in VX_1 are connected to the special vertex called *Sink*. The capacity of connecting edges is equal to the weight of the corresponding vertices in V. For example, the left half of Figure 19 shows the bipartite graph corresponding to the constraint graph of Figure 18. Next the maximum flow in the network X is found, with vertices in VX_2 being source vertices of capacity equal to the weights of the corresponding edges in E. See the right half of Figure 19 (thick edges have non-zero flow).

The maximum flow found in X induces a partition of the original graph G into a partially ordered set of strongly connected components—giving a sequence of solvable subgraphs of G, provided G is well-constrained—as in Figure 18, according to the following rules: if the flow in X from the vertex $z \in VX_2$ that is connected to the vertices $x, y \in VX_1$ is sent toward x, then in the graph G, the edge corresponding to z (between vertices x and y) becomes an oriented edge directed from y to x. If the flow from z is sent toward both x and y, then the edge is bidirected toward both x and y. (Recall that

a strongly connected component S in this directed graph is a subgraph such that for any two vertices $a, b \in S$, there is an oriented path from a to b.) The partial ordering of components is induced as follows. Let K and L be two strongly connected components in the oriented version of G. If all the edges between vertices of K and L are pointing toward L, then L should be simplified after K.

DEFINING THE SIMPLIFIER MAP

We capture the transformations performed by the MM DR-planner described above, by describing its simplifier maps (recall the definitions in Section 3).

Let $G = (V, E)$ be the geometric constraint graph. Denote by $G_1 = (V_1, E_1 = E)$ the directed graph after weights of some vertices have been reduced as described above and a partial ordering of strongly connected components has been found. First $S_1 \subseteq G_1$ is located such that S_1 is strongly connected. Then S_1 is simplified into the vertex v_1 of weight 0. The other vertices of G_1 remain unchanged. Edges of G_1 that had both endpoints outside of S_1 are unchanged, edges that had both endpoints in S_1 are removed, edges that had exactly one endpoint in S_1 have this endpoint replaced by v_1. In the next step, C_2—the next strongly connected component in a topological ordering of the components in $G_2 = T_1(G_1)$—is located. The subgraph S_2 is set to be $\{v_1\} \cup C_2$. Then S_2 is simplified into the vertex v_2 of weight 0. This process is continued until G_k consists of one vertex. The simplifier maps are formally defined as follows.

- $T_i(S_i) = v_i$ where v_i is the vertex in G_{i+1} of weight 0.
- If $B \subseteq G_i, B \cap S_i = \emptyset$, then $T_i(B) = B$.
- If $B \subseteq G_i, B \cap S_i \neq \emptyset$, then the image of B, $T_i(B) = \{v_i\} \cup (B \setminus S_i)$.

REDEFINING SOLVABILITY

CLAIM 4.8. *The MM algorithm is a not a general DR-planner.*

PROOF. While the MM algorithm can correctly classify as solvable and decompose well-constrained and well-overconstrained graphs and correctly classify underconstrained graphs that have no overconstrained subgraphs as being unsolvable, it is unable to correctly classify an underconstrained graph that has an overconstrained subgraph. Consider Figure 21, the weight of all the edges is 1, of the vertices as indicated. Graph $ABCDE$ has density -3, it contains overconstrained subgraphs AC, BC, CD, CE, and removal of say edge AC will result in $ABCDE$ becoming non-dense, hence $ABCDE$ is not well-overconstrained and not solvable. The MM algorithm may or may not detect this, depending on the initial choice of vertices whose weights are to be reduced. Suppose that the weight of the vertex E was reduced by 1 and the weight of the vertex D by 2. Then the corresponding maximum possible flow f and the corresponding decomposition into strongly connected components are shown in Figure 22. Note that it is impossible to simplify the strongly connected component A, and thus there can be no DR-plan for this initial choice of vertices for reducing weights. On the other hand, if the weight of A had been reduced by 2 and weight of D by 1, then the maximum flow is larger than f (in fact, no choice of vertices for weight reduction can give a larger flow) and it can be checked that MM would yield a DR-plan, see Figure 23 ($S_1 = A, S_2 = AC, S_3 = ABC, S_4 = ABCD, S_5 = ABCDE$). □

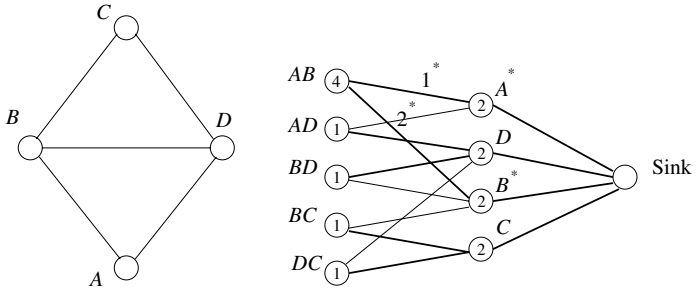

Figure 20. Constraint graph and network flow with three extra flow units at AB.

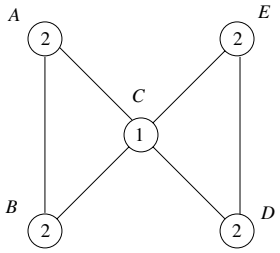

Figure 21. The existence of a DR-plan depends on the initial choice of vertices whose weights are reduced.

Thus MM runs into problems in the presence of overconstrained subgraphs, unless G happens to be well-overconstrained, which cannot be *a priori* checked without relying on an algorithm for detecting overconstrained subgraphs and replacing them by well-constrained ones. Therefore it is necessary and sufficient to locate overconstrained subgraphs and replace them by well-constrained ones, in order to guarantee that MM will work.

The author of Pabon (1993) does *not* specify how to do this. The following modification similar to that of Hendrickson (1992) could be used, as was suggested by a reviewer, and completed here.

First a maximum flow in the unmodified network X (i.e. where vertex weights are not reduced) is found. After that, for every source vertex $v \in VX_2$, except one vertex v_m, the following two steps are repeated.

- Three extra units of flow are sent from v, possibly rearranging existing flows in X.
- These 3 units of flow are removed, without restoring original flows.
- For vertex v_m, 3 units of flow are sent but not removed.

For example, consider Figure 20. The constraint graph is shown on the left, the weights of all vertices are 2, the weights of all edges are 1. The resulting network flow is shown on the right, assuming that AB was the last vertex v_m. The extra three units of flow and their destination vertices A and B are marked by asterisks. This particular flow induces a weight reduction of A by 1 unit and of B by 2 units.

This modification identifies overconstrained subgraphs since it is impossible to send all three extra units from at least one edge of an overconstrained graph. These overcon-

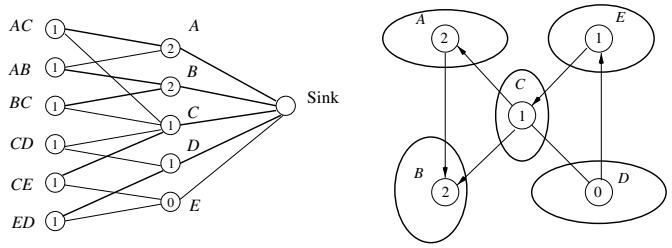

Figure 22. Maximum flow and decomposition given the bad initial choice.

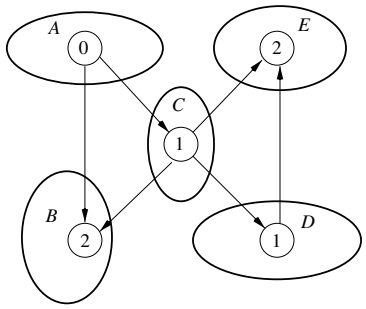

Figure 23. Decomposition given the good initial choice.

strained graphs have to be modified by the designer to become well-constrained, and the flow algorithm is *run again*.

This is *important* because otherwise the DR-algorithm cannot proceed, since in underconstrained graphs with overconstrained subgraphs, strongly connected components do not necessarily correspond to a sequence of solvable subgraphs.

To reflect the modification above, we make the following *concession* for MM, *(MM1)*: the input constraint graph G has no overconstrained subgraphs in G. A subgraph $A \subseteq G$ is "solvable" if it has zero density, after the vertices for weight reduction by D are chosen (these can be chosen arbitrarily, provided there are no overconstrained subgraphs).

PERFORMANCE ANALYSIS

In this section, we analyze the MM algorithm with respect to the various performance measures defined in Section 3.

Note that despite the concession MM1, the MM algorithm has the following drawbacks: the DR-plan is uniquely determined, once vertices whose weights are reduced are chosen. For example, in Figure 20 after the weight of A is reduced by 1 and the weight of B by 2 units, solvable subgraphs to be simplified are $S_1 = \{A, B\}, S_2 = \{T_1(S_1), D\}, S_3 = \{T_2(S_2), C\}$. While the subgraph BCD is also solvable (if say initially weights of vertices B and C were reduced instead of A and B), it cannot be chosen as one of the S_i after the weights of A and B are reduced.

This causes MM to have bad worst-choice and best-choice approximation factors and to be unable to incorporate general designer decomposition.

The following is straightforward from the description of the simplifiers.

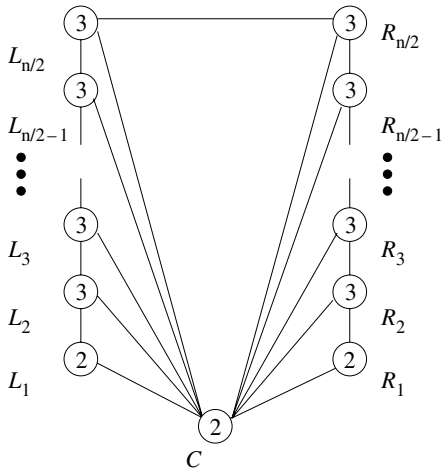

Figure 24. Bad best-choice approximation.

Claim 4.9. *Under the concession MM1, the MM algorithm is a valid DR-planner.*

Claim 4.10. *Under the concession MM1, the MM algorithm is strictly solvability preserving (and therefore solvability preserving).*

Proof. Suppose that a subgraph A of the input graph G is "solvable", i.e. $d(A) = 0$. Let S_i be the "solvable" subgraph to be simplified at the current stage. Let $B = A \cap S_i, C = A \setminus B$. Since we assume that G does not contain any overconstrained subgraphs, $d(B) \leq 0$ and $d(A \cup S_i) \leq 0 \Rightarrow d(A \cup S_i) = d(A) + d(S_i) - d(B) \leq 0 \Rightarrow 0 + 0 - d(B) \leq 0 \Rightarrow d(B) = 0$. Thus $d(T_i(A)) = d(T_i(C)) + d(T_i(B)) = d(C) + 0 = d(A) - d(B) = 0$, therefore $T_i(A)$ is also solvable. □

Claim 4.11. *Under the concession MM1, the MM algorithm is complete.*

Proof. Let A be a proper solvable subgraph of the $S_i, i \geq 2$. Since A is solvable, there cannot be any edges outside of A pointing toward A (this is because there is no room for flows of "outside" edges toward the vertices of A). Recall that S_i is the union of $C_i \cup \{v_{i-1}\}$, where C_i is the first strongly connected component in the topological ordering of the components at the stage i, and v_{i-1} is the simplification of S_{i-1}. However, unless $A = v_{i-1}$, the first strongly connected component at this stage would have been $A \setminus \{v_{i-1}\}$, not C_i, which contradicts the choice of C_i at stage i. □

Claim 4.12. *Under the concession MM1, the best-choice (and worst-choice) approximation factor of MM is at most $O(\frac{1}{n})$.*

Proof. To prove the bound on the best-choice approximation factor consider Figure 24. The left and right columns contain $n/2$ vertices each. The weights of all the vertical edges are 2, the weights of all other edges are 1, the weights of the vertices are as indicated, and the geometry-dependent constant $D = 3$.

Note that all solvable subgraphs in Figure 24 could be divided into three classes. The

first class consists of the subgraphs $CL_1L_2; CL_1L_2L_3; \ldots; CL_1L_2 \ldots L_{n/2-1}L_{n/2}$. The second class consists of the subgraphs $CR_1R_2; CR_1R_2R_3; \ldots; CR_1R_2 \ldots R_{n/2-1}R_{n/2}$. The third class contains the solvable subgraphs that contain both L and R vertices. There is only one element in this class—the entire graph $CL_1L_2 \ldots L_{n/2}R_1R_2 \ldots R_{n/2}$. There is an optimal DR-plan of constant size that takes $S_1 = CL_1L_2, S_2 = S_1 \cup L_3, \ldots, S_{n/2-1} = S_{n/2-2} \cup L_{n/2}$. After that it takes $S_{n/2} = CR_1R_2, S_{n/2+1} = S_{n/2} \cup R_3, \ldots, S_n = S_{n-1} \cup R_{n/2}$. Finally it takes $S_{n+1} = S_{n/2-1} \cup S_n$.

However all DR-plans found by MM will have size $O(n)$. The reason for this is that MM is unable to simplify solvable subgraphs on the left of the Figure 24 independently from the solvable subgraphs on the right. More formally, let S_1 be the first subgraph simplified by MM under some DR-plan Q. If S_1 belongs to the third class of solvable subgraphs, then the size of Q is $O(n)$. Otherwise, without loss of generality we can assume that S_1 belongs to the first class. According to the definition of MM, the simplification of S_1 is a vertex v_1 of weight 0. After this simplification any strongly connected component that contains some R_i should also contain all of the $R_1 \ldots R_{n/2}$. Hence there is an S_i in Q such that $R_1R_2 \ldots R_{n/2} \subset S_i$. Hence the size of Q is $O(n)$. □

Next, we consider three other performance measures discussed in Section 3.

Claim 4.13. *Under the concession MM1, a simple modification of the MM algorithm is able to adapt to underconstrained graphs. This modification, however, increases the complexity by a factor of n.*

Proof. Suppose that the graph G is underconstrained. Consider the maximum flow found by MM in the network X corresponding to G, as described in the section Redefining Solubility. There are two cases.

The first case is when the last vertex v_m in X corresponds to an edge of some solvable subgraph A_1. Then all vertices v_i in X corresponding to vertices of A_1 have their capacities completely filled, since A_1 is solvable. On the other hand, at least one vertex v of G will not have its capacity filled completely, since G is underconstrained. Let W be the set of such vertices v. The new modification of MM would proceed by removing all vertices of X corresponding to vertices of W and edges adjacent to W, as well as flows originating at such edges. Once this is done, a new set W is recomputed and removed, until all remaining vertices of X that represent vertices of G have their capacities filled completely. These vertices comprise a solvable subgraph A_1 such that no supergraph of A_1 is solvable. Once the subgraph A_1 (and its DR-plan Q_1) is found, it could be removed from G and the process applied recursively to $G \setminus A_1$ to find a solvable subgraph A_2, its DR-plan Q_2, etc.

The second case is when the last vertex v_m in X is not contained in any solvable subgraph. Then constructing and removing sets W as described above will completely exhaust G, without finding any solvable graphs. The new modification of MM would proceed by finding another maximum flow in network X corresponding to G such that the last vertex v_m^i is different from v_m. Again two cases are considered for vertex v_m^i. If all $v_m^i \in G$ are not contained in any solvable subgraph, then G does not contain any solvable subgraph and the process terminates.

Note that this modification offhand increases the complexity of MM by a factor of n, and there does not seem to be any obvious way to prevent this factor.

This modification of MM outputs a DR-plan $Q = Q_1, \ldots, Q_k$ for a set of solvable

subgraphs $A_1, \ldots, A_k$ such for any i no supergraph of A_i is solvable and there are no solvable subgraphs $B \subseteq G$ such that $B \cap A_j = \emptyset$, $\forall j$. Thus this modification of MM is able to adapt to underconstrained graphs. □

OBSERVATION 4.14. *(i) Under the concession MM1, MM has the Church–Rosser property since simplifying any S_i that is solvable at the current stage preserves the density of the whole graph G_{i+1} (i.e. G_{i+1} is solvable if and only if G_i is).*

(ii) Under the concession MM1, MM is able to incorporate a design decomposition P specified by the designer if and only if for every $P_k, P_t \in P$ such that $P_k \cap P_t \neq \emptyset$ either $P_k \subseteq P_t$ or $P_t \subseteq P_k$.

Proof is similar to the corresponding proof for the SR algorithm in Claim 4.7.

4.3. COMPARISON OF PERFORMANCE

Next, we give a table comparing the SR and MM DR-planners with respect to the performance measures of Section 3. "Underconstr." refers to the ability to deal with underconstrained graphs, "Design decomposition" refers to the ability to incorporate design decompositions specified by the designer.

Performance measure	SR	MM
Generality	No	Yes†
Underconstr.	No(Yes*)	Yes$^{\dagger,\diamondsuit}$
Design decomposition	No(Yes$^{*,\circ}$)	No(Yes$^{\dagger,\circ}$)
Validity	No(Yes*)	Yes†
Solvability	No(Yes*)	Yes†
Strict solvability	No(Yes*)	Yes†
Complete	No(No*)	Yes†
Worst approximation factor	0 ($O(\frac{1}{n})^{*}$)	$O(\frac{1}{n})^{\dagger}$
Best approximation factor	0 ($O(\frac{1}{n})^{*}$)	$O(\frac{1}{n})^{\dagger}$
Church–Rosser	No(Yes*)	Yes†
Complexity	$O((m+n)^2)$	$O(n(m+n))^{\dagger}$

The superscript "*" refers to a narrow class of DR-plans: those that require the solvable subsystems S_i to be based on triangles or a fixed repertoire of patterns. The superscript "†" refers to results that were not true for the original MM algorithm developed by Ait-Aoudia *et al.* (1993) and Pabon (1993) and proved in this paper through a modification of MM described in Section 4.2. The superscript $\diamondsuit$ refers to a restricted class of graphs in which there are no overconstrained subgraphs. It also refers to a further modification of MM described in Claim 4.13, which however, increases the complexity by *a factor of* n. The superscript "$\circ$" refers to strong restrictions on the design decompositions that can be incorporated into DR-plans by SR and MM.

Acknowledgements

We thank two anonymous reviewers for their careful reading of the paper, and one of them for suggesting the modification which we completed and which significantly improves the MM algorithm, as described in Section 4.2.

References

Ait-Aoudia, S., Jegou, R., Michelucci, D. (1993). Reduction of constraint systems. In *Compugraphics*, pp. 83–92. Alvor, Portugal.

Blum, L., Shub, M., Smale, S. (1989). On a theory of computation and complexity over the real numbers: NP-completeness, recursive functions and universal machines. *Bull. Am. Math. Soc.*, **21**, 1–46.

Bouma, W., Fudos, I., Hoffmann, C., Cai, J., Paige, R. (1995). A geometric constraint solver. *Comput. Aided Des.*, **27**, 487–501.

Bronsvoort, W. F., Jansen, F. W. (1994). Multiview feature modeling for design and assembly. In Shah, J., Mantyla, M., Nau, D. S. eds, *Advances in Feature Based Modeling*, pp. 315–330. Amsterdam, Elsevier Science.

Canny, J. (1993). Improved algorithms for sign determination and existential quantifier elimination. *Comput. J.*, **36**, 409–418.

Canny, J., Emiris, I. (1993). An efficient algorithm for the sparse mixed resultant. In Cohen, G., Mora, T., Moreno, O. eds, *Proceedings of the 10th International Symposium on Applied Algebra, Algebraic Algorithms, and Error Correcting Codes*, LNCS **263**, pp. 89–104. Berlin, Springer-Verlag.

Chou, S. C., Gao, X. S., Zhang, J. Z. (1996). A method of solving geometric constraints. Technical Report, Wichita State University, Department of Computer Science.

Collins, G. (1975). Quantifier elimination for real closed fields by cylindrical algebraic decomposition, LNCS **33**, pp. 134–183. Berlin, Springer-Verlag.

Cox, D., Little, J., O'Shea, D. (1998). *Using Algebraic Geometry*. New York, Springer-Verlag.

Crippen, G., Havel, T. (1988). *Distance Geometry and Molecular Conformation*. New York, John Wiley & Sons.

Durand, C. (1998). Symbolic and numerical techniques for constraint solving. Ph.D. Thesis, Purdue University, Computer Science Department.

Fang, S. (1992). Robustness in geometric modeling. Ph.D. Thesis, University of Utah.

Fudos, I. (1995). Geometric constraint solving. Ph.D. Thesis, Purdue University, Department of Computer Science.

Fudos, I., Hoffmann, C. M. (1996a). Constraint-based parametric conics for CAD. *Comput. Aided Des.*, **28**, 91–100.

Fudos, I., Hoffmann, C. M. (1996b). Correctness proof of a geometric constraint solver. *Int. J. Comput. Geom. Appl.*, **6**, 405–420.

Fudos, I., Hoffmann, C. M. (1997). A Graph-Constructive Approach to Solving Systems of Geometric Constraints. *ACM Trans. Graph.*, **16**, 179–216.

Gabow, H., Westermann, H. (1988). Forests, frames and games: Algorithms for matroid sums and applications. In *Proceedings of the Twentieth Annual ACM Symposium on the Theory of Computing, Chicago, Illinois, U.S.A., 2–4 May 1998*, pp. 407–421. New York, ACM Press.

Gao, X. S., Chou, S. C. (1998a). Solving geometric constraint systems. I. A global propagation approach. *Comput. Aided Des.*, **30**, 47–54.

Gao, X. S., Chou, S. C. (1998b). Solving geometric constraint systems. II. A symbolic approach and decision of RC-constructibility. *Comput. Aided Des.*, **30**, 115–122.

Graver, J., Servatius, B., Servatius, H. (1993). *Combinatorial Rigidity, Graduate Studies in Mathematics*.

Grigor'ev, D. Y., Vorobjov, N. N. (1988). Solving systems of polynomial inequalities in subexponential time. *J. Symb. Comput.*, **5**, 37–64.

Hastad, J. (1996). Clique is hard to approximate within $n^{1-\epsilon}$. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, pp. 627–636. Burlington, IEEE press.

Havel, T. (1991). Some examples of the use of distances as coordinates for Euclidean geometry. *J. Symb. Comput.*, **11**, 579–594.

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

Hoffmann, C. M. (1997). Solid modeling. In Goodman, J. E., O'Rourke, J. eds, *CRC Handbook on Discrete and Computational Geometry*. Boca Raton, FL, CRC Press.

Hoffmann, C. M., Joan-Arinyo, R. (1998). CAD and the product master model. *Comput. Aided Des.*, **30**, 905–919.

Hoffmann, C. M., Lomonosov, A., Sitharam, M. (1997). Finding solvable subsets of constraint graphs. In *Proceedings of Principles and Practice of Constraint Programming '97, Linz, Austria, LNCS* **1330**, pp. 463–477. Berlin, Springer-Verlag.

Hoffmann, C. M., Lomonosov, A., Sitharam, M. (1998). Geometric constraint decomposition. In Bruderlin, Roller eds, *Geometric Constraint Solving*. Berlin, Springer-Verlag.

Hoffmann, C. M., Peters, J. (1995). Geometric constraints for CAGD. In Daehlen, M., Lyche, T., Schumaker, L. eds, *Mathematical Methods for Curves and Surfaces*, pp. 237–254. Vanderbilt University Press.

Hoffmann, C. M., Rossignac, J. (1996). A road map to solid modeling. *IEEE Trans. Vis. Comput. Graphics*, **2**, 3–10.

Hoffmann, C. M., Vermeer, P. J. (1994). Geometric constraint solving in R^2 and R^3. In Du, D. Z., Hwang, F. eds, *Computing in Euclidean Geometry*, 2nd edn. Singapore, World Scientific Publishing.

Hoffmann, C. M., Vermeer, P. J. (1995). A spatial constraint problem. In *Workshop on Computational Kinematics, France*. Sophia-Antipolis, INRIA.
Hopcroft, J. E., Tarjan, R. E. (1973). Dividing a graph into triconnected components. *SIAM J. Comput.*, **2**, 135–158.
Hsu, C. (1996). Graph-based approach for solving geometric constraint problems. Ph.D. Thesis, University of Utah, Department of Computer Science.
Imai, H. (1985). On combinatorial structures of line drawings of polyhedra. *Discrete Appl. Math.*, **10**, 79–92.
Itai, A., Rodeh, M. (1978). Finding a minimum circuit in a graph. *SIAM J. Comput.*, **4**, 413–423.
Khovanskii, A. G. (1978). Newton polyhedra and the genus of complete intersections. *Funktsional'nyi Analiz i Ego Prilozheniya*, **12**, 51–61.
Klein, R. (1996). Geometry and feature representation for an integration with knowledge based systems. In *Geometric Modeling and CAD*. Florida, U.S.A., Chapman-Hall.
Klein, R. (1998). The role of constraints in geometric modeling. In Bruderlin, Roller eds, *Geometric Constraint Solving and Applications*. Berlin, Springer-Verlag.
Kraker, K. J., Dohmen, M., Bronsvoort, W. F. (1997). Maintaining multiple views in feature modeling. In *ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pp. 123–130. New York, ACM press.
Kramer, G. (1992). *Solving Geometric Constraint Systems*. Cambridge, MA, U.S.A., MIT Press.
Laman, G. (1970). On graphs and rigidity of plane skeletal structures. *J. Eng. Math.*, **4**, 331–340.
Latham, R., Middleditch, A. (1996). Connectivity analysis: a tool for processing geometric constraints. *Comput. Aided Des.*, **28**, 917–928.
Lazard, D. (1981). Résolution des systèmes d'équations algébriques. *Theor. Comput. Sci.*, **15**, 77–110.
Lazard, D. (1991). A new method for solving algebraic systems of positive dimension. *Discrete Appl. Math.*, **33**, 147–160.
Mantyla, M., Opas, J., Puhakka, J. (1989). Generative process planning of prismatic parts by feature relaxation. In *Advances in Design Automation, Computer Aided and Computational Design*, pp. 49–60. Sacramento, CA, U.S.A., ASME.
Middleditch, A., Reade, C. (1997). A kernel for geometric features. In *ACM/SIGGRAPH Symposium on Solid Modeling Foundations and CAD/CAM Applications*. New York, ACM press.
Newell, M. E., Evans, D. C. (1976). Modeling by computer. In *IFIP Working Conference on CAD Systems, Austin, TX*, pp. 291–297. Amsterdam, North-Holland.
Owen, J. (1991). Algebraic solution for geometry from dimensional constraints. In *ACM Symposium Found. of Solid Modeling, Austin, TX*, pp. 397–407. ACM.
Owen, J. (1996). Constraints on simple geometry in two and three dimensions. *Int. J. Comput. Geom. Appl.*, **6**, 421–434.
Pabon, J. A. (1993). Modeling method for sorting dependencies among geometric entities. *United States Patent 5,251,290*.
Renegar, J. (1992). On the computational complexity and the first order theory of the reals, Part I. *J. Symb. Comput.*, **13**, 255–299.
Ruiz, O. E., Ferreira, P. M. (1996). Algebraic geometry and group theory in geometric constraint satisfaction for computer-aided design and assembly planning. *IIE Trans. Des. Manuf.*, **28**, 281–294.
Saliola, F., Whiteley, W. (1999). Constraint configurations in CAD: circles, lines and angles in the plane. Preprint, York University.
Semenkov, O. I. (1976). An experimental CAD/CM system. In *3rd International IFIP/IFAC Conference on Programming Languages for Machine Tools, Stirling, Scotland*, pp. 397–403. Amsterdam, North-Holland.
Serrano, D. (1990). Managing constraints in concurrent design: first steps. In *Proceedings of Computers in Engineering, Boston, MA, 1990*, pp. 159–164.
Serrano, D., Gossard, D. C. (1986). Combining mathematical models with geometric models in CAE systems. *Computers in Engineering, Chicago, IL. ASME*, **1**, 277–284.
Sridhar, N., Aggarwal, R., Kinzel, G. L. (1993). Active occurence matrix based approach to design decomposition. *Comput. Aided Des.*, **25**, 500–512.
Sridhar, N., Aggarwal, R., Kinzel, G. L. (1996). Algorithms for the structural diagnosis and decomposition of sparse, underconstrained, systems. *Comput. Aided Des.*, **28**, 237–249.
Sturmfels, B. (1993). Sparse elimination theory. In *Proceedings of the Computational Algebraic Geometry and Commutative Algebra*, pp. 377–396. Cambridge, Cambridge University Press.
Sugihara, K. (1985). Detection of structural inconsistency in systems of equations with degrees of freedom and its applications. *Discrete Appl. Math.*, **10**, 297–312.
Tay, T. (1999). On the generic rigidity of bar frameworks. *Adv. Appl. Math.*, **23**, 14–28.
Tay, T., Whiteley, W. (1985). Generating isostatic frameworks. *Topologie Structurale*, **11**, 21–69.
Wang, D. (1993). An elimination method for polynomial systems. *J. Symb. Comput.*, **16**, 83–114.
Whiteley, W. (1992). Matroids and rigid structures. In Matroid Applications, *volume 40 of Encyclopedia Math. Appl.*, pp. 1–53. Cambridge, Cambridge University Press.

Whiteley, W. (1997). Rigidity and scene analysis. In *Handbook of Discrete and Computational Geometry*, pp. 893–916. Florida, U.S.A., CRC Press.

Originally Received 17 February 1998
Accepted 11 June 2000