Artificial Intelligence 58 (1992) 327–360
Elsevier

ARTINT 957

A geometric constraint engine

Glenn A. Kramer

Schlumberger Laboratory for Computer Science, 8311 RR 620 North, P.O. Box 200015, Austin, TX 78720-0015, USA

Abstract

Kramer, G.A., A geometric constraint engine, Artificial Intelligence 58 (1992) 327–360.

This paper describes a geometric constraint engine for finding the configurations of a collection of geometric entities that satisfy a set of geometric constraints. This task is traditionally performed by reformulating the geometry and constraints as algebraic equations which are then solved symbolically or numerically. Symbolic algebraic solution is NP-complete. Numerical solution methods are characterized by slow runtimes, numerical instabilities, and difficulty in handling redundant constraints. Many geometric constraint problems can be solved by reasoning symbolically about the geometric entities themselves using a new technique called *degrees of freedom analysis.* In this approach, a plan of measurements and actions is devised to satisfy each constraint incrementally, thus monotonically decreasing the system's remaining degrees of freedom. This plan is used to solve, in a maximally decoupled form, the equations resulting from an algebraic representation of the problem. Degrees of freedom analysis results in a polynomial-time, numerically stable algorithm for geometric constraint satisfaction. Empirical comparison with a state-of-the-art numerical solver in the domain of kinematic simulation shows degrees of freedom analysis to be more robust and substantially more efficient.

1. Introduction

Geometric reasoning plays a fundamental role in our understanding of the physical world. An important task in geometric reasoning is the geometric constraint satisfaction problem (GCSP): Given a collection of geometric entities, or *geoms,* and constraints that describe how the geoms interact with each other, find their positions, orientations, and dimensions so as to satisfy all constraints simultaneously. Solving GCSPs is central to several related domains: describing mechanical assemblies, constraint-based sketching and design, geometric modeling for computer-aided design, and kinematic analysis of robots and other mechanisms.

Correspondence to: G.A. Kramer, Schlumberger Laboratory for Computer Science, 8311 RR 620 North, P.O. Box 200015, Austin, TX 78720-0015, USA. Telephone: (512) 331-3715. E-mail: gak@slcs.slb.com.

General-purpose constraint satisfaction techniques are not well suited to the solution of constraint problems involving complicated geometry, for reasons to be explained shortly. This paper describes a novel technique, called *degrees of freedom analysis,* for solving GCSPs. It avoids a number of drawbacks associated with traditional approaches to solving GCSPs. Degrees of freedom analysis borrows from techniques originally developed for the analysis and synthesis of mechanical devices [6]. These techniques have been formalized and generalized so that they apply to a wider class of geometric constraint satisfaction problems.

Existing programs which solve GCSPs represent geoms and constraints as algebraic equations, whose real solutions yield the numerical values describing the positions, orientations, and dimensions of the geoms. Such equation sets are highly nonlinear and highly coupled, and in the general case require iterative numerical solution techniques. Iterative numerical programs are not particularly efficient, and can have problems with stability and robustness [18]. For many tasks (e.g., simulation and optimization of mechanical devices) the same equations are solved repeatedly, which makes a “hard-wired”, or compiled, solution desirable.

In theory, symbolic analysis of the equations can often yield a non-iterative, closed-form solution, or can help reduce the number of redundant generalized coordinates in an iterative problem. Once found and compiled, such a closed-form solution may be executed in time nearly linearly proportional to the size of the constraint problem. However, the computational intractability of symbolic algebraic solution of the equations renders this approach impractical [11].

Degrees of freedom analysis solves GCSPs by reasoning symbolically about geometry, rather than equations, leading to more efficient algorithms. Degrees of freedom analysis uses two models of a constraint problem: a symbolic geometric model and a detailed numerical model. The geometric model is used to reason symbolically about how to assemble the geoms so as to satisfy the constraints incrementally. The “assembly plan” thus developed is then used to guide the solution of the complex nonlinear equations—derived from the numerical model—in a highly decoupled, stylized manner. This approach allows finding non-iterative, closed-form solutions to GCSPs whenever possible, and allows formulating iterative problems with a minimal number of redundant generalized coordinates when closed-form solutions do not exist.

Degrees of freedom analysis was developed for analyzing problems of rigid-body kinematics, and was tested with an implemented computer program called The Linkage Assistant (TLA) [9,10]. This paper describes extensions and alterations to degrees of freedom analysis to cover a broader range of geometric entities and constraints. A program called the Geometric Constraint

Engine (GCE) has been developed to test these extensions. All examples in this paper have been solved by GCE.

1.1. Domain

Design and analysis of physical systems often require a representation of the geometry of the system. While some problems such as finite element analysis [5] and design using deformable surfaces [4] are inherently iterative in nature, other problems can, in principle, be treated either entirely or in large part using closed-form solution techniques. In practice, many such problems are still treated iteratively, either due to the complexity of deriving a direct formulation, or due to the use of more general solution techniques that also handle the (possibly small) portions of the problem which require iteration.

Domains such as constraint-based sketching and mechanical part design fall into this latter category. They tend to rely on complex combinations of relatively simple geometric elements, such as points, lines, and circles, and a small collection of constraints such as coincidence, tangency, and parallelism. For example, profile design for mechanical devices involves defining a closed perimeter curve, usually comprised of line segments and arcs, with a set of features, such as holes and slots, in the interior. In three dimensions, collections of simple solids (e.g., spheres, cones, cylinders) are combined to yield a solid structure. While the positions of the geoms in such structures often may be computed in a closed-form, analytic manner, the sequence of transformational operations required to satisfy the constraints may be quite complex. In the past, the designer of the part had to provide the transformation sequences [19]. Degrees of freedom analysis generates such sequences of transformations automatically.

Texts in fields such as mechanical engineering or computer-aided design employ simple examples using algebraic techniques inspired by, and grounded in, the geometric nature of the problems being analyzed. Kinematic analysis of rigid-body mechanisms is an example in which geometric construction techniques are used [6]. However, real-world codes for kinematic analysis bear no resemblance to the human problem solving techniques outlined in such texts, and are quite unintuitive. Degrees of freedom analysis leads to a more understandable way of solving these problems by automatically generating the geometric constructions required for analysis.

1.2. Terminology

The objects of interest in solving GCSPs will be called *geoms*. Some examples of geoms are lines, line segments, circles, and rigid bodies. Geoms have degrees of freedom, which allow them to vary in location or size.

For example, in 3D space, a rigid body has three translational and three rotational degrees of freedom. A circle with a variable radius has three translational, two rotational, and one dimensional degree of freedom (a third rotational degree of freedom is not required because the circle is invariant under rotation about its axis).

The *configuration variables* of a geometric object are defined as the minimal number of real-valued parameters[1] required to specify the object in space unambiguously. The configuration variables parameterize an object s translational, rotational, and dimensional degrees of freedom (DOFs), with one variable required for each DOF. A *configuration* of a geom is a particular assignment of the configuration variables, yielding a unique instantiation of the geom.

Using this terminology, a GCSP is defined as follows: Given a set of geoms and constraints between them, find the configurations of the geoms such that all constraints are satisfied. The collection of entities and constraints is called the *constraint system*, or simply the *system*.

2. Degrees of freedom analysis

Degrees of freedom analysis shares much with a body of principles found in texts on the graphical analysis of mechanisms. In fact, the earliest analyses of mechanisms were entirely graphical (i.e., geometrical) in nature. As algebraic methods were developed, the graphical methods were abandoned due to the error inherent in such manual approaches. But the algebraic techniques are hardly intuitive; therefore, the graphical methods are still significant. They "maintain touch with physical reality to a much greater degree than do the algebraic methods" and "serve as useful guides in directing the course of equations" [6, p. 215]. Degrees of freedom analysis encapsulates this "intuition" in a formal method.

One way to characterize degrees of freedom analysis is as a forward chaining system performing geometric constructions to ascertain the location of the various geoms. In geometry theorem proving, forward chaining is infeasible because the space of possible inference is infinite [13]. In degrees of freedom analysis, each geometric construction (comprised of a sequence of measurements and actions) satisfies some constraint, but also reduces the number of degrees of freedom in the composite system of geoms. Eventually, all degrees of freedom are consumed by actions, and the inference process terminates. Thus, forward chaining is feasible.

[1]Also known as "generalized coordinates" [15].

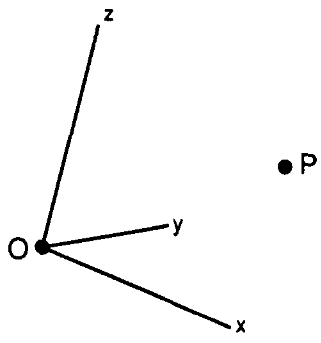

Fig. 1. A rigid body with two embedded points.

2.1. Resources, measurements, and actions

Solving GCSPs using degrees of freedom analysis relies on a representation shift from reasoning about the real-valued configuration variables to reasoning about the DOFs of the actual geometric entities. The equations that relate configuration variables to each other may be complicated, tightly coupled, and highly nonlinear; in addition, the domains of the configuration variables are continuous, yielding an infinite search space. In contrast, an object's degrees of freedom form a compact, discrete-valued description of the state of the object.

Degrees of freedom form abstract equivalence classes describing the state of a geometric entity without specifying how the constraints that lead to that state are satisfied. DOFs may be considered resources which are consumed by "physically" moving geoms to satisfy constraints. Further actions are then confined to those that do not violate any previously-satisfied constraints. Therefore, every constraint, upon being satisfied, requires that certain quantities be treated as invariant in the satisfaction of subsequent constraints, thereby restricting some number of degrees of freedom. These geometric invariants are represented explicitly.

Reasoning about DOFs is essential to decoupling the constraints. Consider the xyz coordinate frame in Fig. 1, with points $\boldsymbol{O}$, at the origin, and $\boldsymbol{P}$, in some arbitrary location, rigidly fixed in the coordinate frame. As a rigid body, the coordinate frame is parameterized by six configuration variables, three for the translational DOFs (x, y, z) and three for the rotational DOFs (θ, ϕ, ψ).[2] Thus, the coordinate frame is free to translate and rotate in space.

Fixing the position of either point $\boldsymbol{O}$ or $\boldsymbol{P}$ (through the satisfaction of some constraint) removes the three translational DOFs in the system; the coordinate frame may only rotate about the fixed point in order to

[2]In this example, the rotational DOFs are represented using Euler angles.

satisfy subsequent constraints. But consider the constraints in terms of configuration variables. Fixing the position of point $\boldsymbol{O}$ uniquely determines the three translational coordinates:

$$x = x_O,$$

$$y = y_O,$$

$$z = z_O,$$

where $[x_O, y_O, z_O]$ denotes the position of point $\boldsymbol{O}$ in the global reference frame.

In contrast, fixing the position of $\boldsymbol{P}$ (instead of $\boldsymbol{O}$) introduces nonlinear constraint equations into the system to relate the configuration variables to the distance $\overline{\boldsymbol{OP}}$:

$$(x - x_{\boldsymbol{P}})^2 + (y - y_{\boldsymbol{P}})^2 + (z - z_{\boldsymbol{P}})^2 = \overline{\boldsymbol{OP}}^2,$$

$$\tan\theta = (y - y_{\boldsymbol{P}})/(x - x_{\boldsymbol{P}}),$$

$$\tan\phi = [(y - y_{\boldsymbol{P}})/(z - z_{\boldsymbol{P}})]\csc\theta.$$

Solving constraint systems algebraically is difficult because of this type of coupling between configuration variables. The coupling is entirely an artifact of the way in which the system is modeled; for example, if the same rigid body is modeled with the coordinate frame centered at point $\boldsymbol{P}$, then satisfying a constraint involving point $\boldsymbol{O}$ leads to coupled equations.

Using incremental movement as a constraint satisfaction scheme, the constraint that point $\boldsymbol{O}$ of the body be at a specific point in space is satisfied by measuring the vector from $\boldsymbol{O}$ to that point, and translating the body by that vector. There is no need to use the local coordinate frame representation, as long as the global position of $\boldsymbol{O}$ can be found by some means. Thus, the identical solution strategy works for point $\boldsymbol{P}$.

Solving in DOF space is simpler because the actions can be specified independently of how the system is parameterized in terms of configuration variables. The action of translating a geom to bring a specific point ($\boldsymbol{O}$ or $\boldsymbol{P}$) to a particular location is independent of the detailed mathematical representation of the geom. The operational semantics shields a constraint satisfaction algorithm from having to know anything about "arbitrary" internal representations.

2.2. The metaphor of incremental assembly

Degrees of freedom analysis employs the notion of *incremental assembly* as a metaphor for solving geometric constraint systems. This use of assembly should not be confused with physical interpretations of assembly as in, for example, robotics applications. In a metaphorical assembly, no physical meaning is ascribed to how the objects move from where they are to where they need to be, a factor which is quite important in a real assembly problem. In solving GCSPs, the values of the geoms' configuration variables constitute the desired answer, rather than the history of how they were calculated.

In a metaphorical assembly, geoms are treated as "ghost objects" which can pass through each other. It is therefore possible to ignore the physical constraints imposed by the boundaries of physical bodies, and instead be concerned only with purely geometric relations. The constraints between "ghost" geoms may be satisfied incrementally; no part is ever moved in a way which violates previously satisfied constraints.

In some real-world problems, like kinematic analysis or profile sketching in computer-aided design, the starting locations of the geoms and their movement toward a configuration which satisfies the constraints is of no concern. What is desired is the globally consistent locations of the geoms. In other domains, such as "real" assembly planning, the "ghost object" metaphor is clearly incorrect. However, real assembly planning can benefit from knowing the final locations of the assembled objects. Disassembling the collection of assembled objects is an easier problem than generating a physically-realizable assembly plan; the disassembly plan can then be run in reverse to create an assembly plan which takes into account the physical constraints [21].

2.3. MAPs and equation solution

A plan (a sequence of measurements and actions) for moving a set of "ghost" geoms from arbitrary configurations to ones satisfying the constraints is called a *metaphorical assembly plan,* or MAP. The generation of a MAP is a problem in symbolic geometry. The sequence of measurements and actions is determined without regard to the actual metric values of the parts.[3] The MAP describes the general form of a solution to a constraint problem. However, symbolic geometry alone is not sufficient to obtain the real values of the configuration variables describing each geom in a system.

To obtain values for configuration variables, degrees of freedom analysis requires a detailed numerical model of each geom. Relating the numerical model to the symbolic model requires a set of operators for translating, rotating, and scaling geoms, and a set of functions that can measure, relative to a global coordinate system, points and vectors embedded in any geom. These capabilities are provided by homogeneous coordinate transforms which most graphics and robotics systems use. The use of the operators allows the solution to the constraint problem to be found in a manner that is independent of the way in which the system is modeled at the detailed numerical level.

[3]As will be seen shortly, geometric degeneracies must be accommodated.

Table 1
Constraints used in GCE.

Constraint name	Explanation
dist:point-point(G_1, G_2, d)	Distance between point G_1 and point G_2 is d.
dist:point-line(G_{pt}, G_ℓ, d)	Distance between point G_{pt} and line G_ℓ is d.
dist:point-plane(G_{pt}, G_{pl}, d)	Distance between point G_{pt} and plane G_{pl} is d.
dist:line-circle(G_ℓ, G_c, d)	Distance between line G_ℓ and circle G_c is d. [a]
angle:vec-vec(G_1, G_2, α)	Angle between vector G_1 and vector G_2 is α.

[a] In two dimensions, $d = 0$ represents a tangency constraint.

3. Representation

3.1. Geometric entities

Geoms can be nested hierarchically in a part–whole relationship; the terms *subgeom* and *parent geom* are used to denote relative position in the hierarchy. *Aggregate* geoms are composed of combinations of *primitive* geoms—points, vectors, and dimensions. A set of measurement, or query, operators allows finding the positions and orientations of points and vectors in the global, or world, coordinate frame.

3.2. Constraints

With the exception of dimensional constraints, all constraints considered here are binary constraints—they relate two geoms. These constraints may additionally involve real parameters. Some examples of constraints used in this paper are shown in Table 1. Dimensional constraints are unary; they relate one geom to a real-valued dimension parameter.

Constraints may apply to subgeoms of a given geom. For example, to constrain two lines to be parallel, one constrains the vectors of those lines to have an angle of zero.

3.3. Invariants

In the TLA system, geometric invariants were stored as arguments to predicates indicating the translational DOFs (TDOFs) and rotational DOFs (RDOFs) of the rigid-body geoms [10]. This scheme works well for the kinematics domain, but does not always work well for other rigid-body systems or for describing geoms with dimensional DOFs.

A rigid-body geom cannot always be characterized by well-defined combinations of TDOFs and RDOFs. In some situations the degrees of freedom are coupled in ways which cannot be divided neatly into TDOFs and RDOFs.

Consider a rigid body $\boldsymbol{B}$ with two points, $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$. Let $\boldsymbol{p}_1$ be constrained to lie in a plane, using the **dist:point-plane** constraint. Let $\boldsymbol{p}_2$ be constrained to lie on a line by a **dist:point-line** constraint. Then $\boldsymbol{B}$ has three degrees of freedom. But $\boldsymbol{B}$'s degrees of freedom cannot be neatly divided into TDOFs and RDOFs, as is now shown.

Let s be the tuple of TDOFs and RDOFs remaining for $\boldsymbol{B}$ after these two constraints have been applied. Now consider the case where $\boldsymbol{p}_1$ is fixed in the plane by satisfying yet another constraint. Then the new tuple of TDOFs and RDOFs, s', is $\langle 0 \text{ TDOF}, 1 \text{ RDOF}\rangle$ ($\boldsymbol{B}$ may rotate about the line connecting $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$). This would suggest that the original s was $\langle 2 \text{ TDOF}, 1 \text{ RDOF}\rangle$, since only translational DOFs were removed by the new constraint. But consider instead the case where the translation of $\boldsymbol{p}_2$ along the line is fixed by a new constraint. Then $s' = \langle 0 \text{ TDOF}, 2 \text{ RDOF}\rangle$ ($\boldsymbol{B}$ may rotate so that $\boldsymbol{p}_1$ remains on a circle in the plane, and it may also rotate about the line connecting $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$). This would suggest that the original s was $\langle 1 \text{ TDOF}, 2 \text{ RDOF}\rangle$. Thus, depending on subsequent constraints, the degrees of freedom in s decompose into differing numbers of TDOFs and RDOFs, making it an ambiguous representation.

A more general approach to representing the degrees of freedom of a geom is to create a data structure that explicitly represents the invariants without assigning them to particular categories (e.g., TDOF or RDOF). In the expanded theory of degrees of freedom analysis, the invariants associated with each geom are stored in a structure called the *invariants record,* which contains several lists of points, vectors, or tuples. The invariants record representation has the advantage over the predicate-based representation of being easily extensible for new constraint types and for different geom types. This data structure is implemented in GCE.

The structure of the invariants record is shown in Table 2. In the table, $\boldsymbol{p}$ represents a point, $\boldsymbol{v}$ a vector, $\mathcal{L}^1$ a one-dimensional locus (e.g., circle, line, parabola), $\mathcal{L}^2$ a two-dimensional locus (e.g., sphere, hyperboloid), $\mathcal{D}$ a dimension, $v_{\mathbb{R}}$ a real value, and G an aggregate geom.

The "invariant points" slot of the invariants record is a list of all points embedded in the geom whose positions are invariant. The "1D-constrained points" slot is a list of ⟨point, locus⟩ tuples denoting those points embedded in the geom which are constrained to lie on one-dimensional loci (similarly for the "2D-constrained points" slot). Vectors, being two-dimensional, can be invariant, or can be constrained to one-dimensional loci (on a unit sphere). Invariant dimensions are those which have been constrained to fixed values.

The last three entries in the invariants record are placeholders for relationships that will later constrain dimensions. Their use is illustrated in Section 4.2.

Table 2
Structure of the invariants record.

Slot	Representation
Invariant points	p
1D-constrained points	$\langle p, \mathcal{L}^1 \rangle$
2D-constrained points	$\langle p, \mathcal{L}^2 \rangle$
Invariant vectors	v
1D-constrained vectors	$\langle v, \mathcal{L}^1 \rangle$
Invariant dimensions	$\langle \mathcal{D}, v_{\mathbb{R}} \rangle$
Fixed-distance points	$\langle p, v_{\mathbb{R}} \rangle$
Fixed-distance lines	$\langle G_\ell, v_{\mathbb{R}} \rangle$
Fixed-distance planes	$\langle G_{\mathrm{pl}}, v_{\mathbb{R}} \rangle$

The cardinalities of the lists in the invariants record at any given stage of the solution process form an *invariants signature.* This signature may be used, along with the type of an as-yet-unsolved constraint, to determine the sequence of measurements and actions which will satisfy that constraint. The invariants signature is represented as a vector of integers, which when read left to right, correspond to the cardinalities of the invariants record slots as described in Table 2. For example, IR[100_10_1_000] describes a geom that has one invariant point, one invariant vector, and one invariant dimension. The underscores separate the signature into point invariants, vector invariants, dimension invariants, and fixed-distance invariants for ease of reading.

The number of DOFs remaining on a partially-constrained geom is calculated by subtracting the number of degrees of freedom restricted by the invariants (an example of this type of calculation appears in Section 4.2.3). If the number of DOFs of a geom becomes zero, the geom is said to be *grounded,* or *fixed.*

4. Action and locus analysis

The two fundamental types of reasoning carried out by degrees of freedom analysis are called *action analysis* and *locus analysis.* They are described through the use of examples. Each example will illustrate the steps used to solve the problem by following the actions of GCE, which implements degrees of freedom analysis.

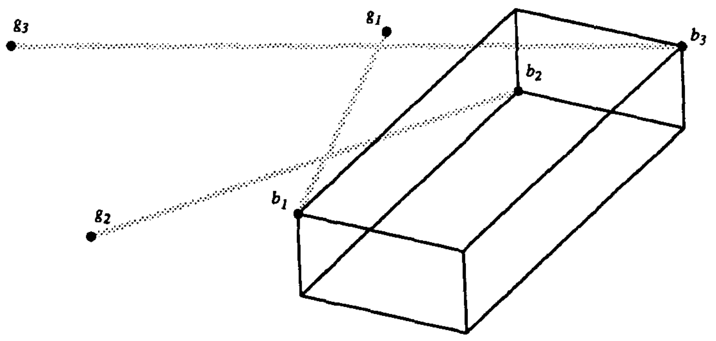

Fig. 2. Brick with three **dist:point-point** constraints.

4.1. Example 1: the brick

Consider a "brick" (a rigid-body geom) with three distinguished points[4] $\boldsymbol{b}_1$, $\boldsymbol{b}_2$, and $\boldsymbol{b}_3$. Another three points, $\boldsymbol{g}_1$, $\boldsymbol{g}_2$, and $\boldsymbol{g}_3$, are fixed in the global coordinate system. The problem is to find a configuration for the brick such that $\boldsymbol{b}_1$ is coincident with $\boldsymbol{g}_1$, $\boldsymbol{b}_2$ with $\boldsymbol{g}_2$, and $\boldsymbol{b}_3$ with $\boldsymbol{g}_3$. The constraints to be satisfied are:

$$\textbf{dist:point-point}(\boldsymbol{g}_1, \boldsymbol{b}_1, 0),$$

$$\textbf{dist:point-point}(\boldsymbol{g}_2, \boldsymbol{b}_2, 0),$$

$$\textbf{dist:point-point}(\boldsymbol{g}_3, \boldsymbol{b}_3, 0).$$

These constraints are depicted graphically in Fig. 2. In this figure, the brick is in some arbitrary configuration, and it must be configured so that the three **dist:point-point** constraints (denoted by gray lines between the points) are all satisfied.

4.1.1. Action analysis

At each step in solving for a geom's configuration, degrees of freedom analysis searches for constraints in which one of the geoms is "fixed enough" so that the other geom can be moved to satisfied the constraint. For example, if one geom of a **dist:point-point** constraint has invariant position, it is fixed enough for the other geom to be moved to satisfy the constraint. If neither geom is fixed, then it makes sense to delay the satisfaction of the constraint, since both geoms might need to be moved subsequently. The process of finding and satisfying constraints using the above strategy is called *action analysis.*

[4]The shape of this "brick" is not important to degrees of freedom analysis. The important information is that this rigid-body geom contains three distinguished points.

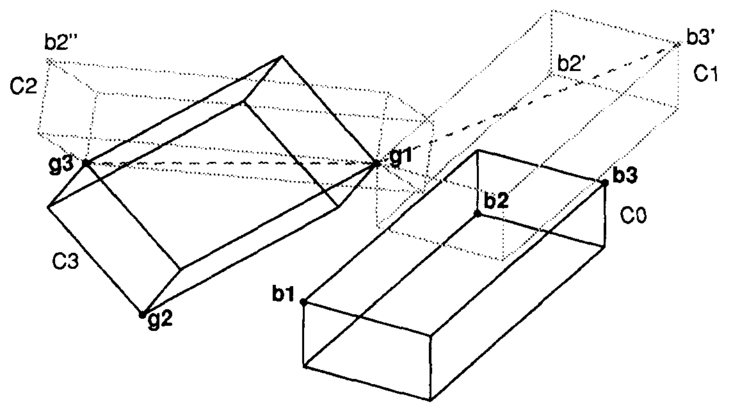

Fig. 3. Brick solution using a geometric approacn.

A geom need only be "fixed enough" to allow the constraint to be satisfied; it need not be grounded. For example, if a line segment $\boldsymbol{L}$ has a fixed orientation, and one endpoint is constrained to a line parallel to the orientation of $\boldsymbol{L}$, then $\boldsymbol{L}$ is "fixed enough" to allow a point to be moved to satisfy a **dist:point-line** constraint.[5]

The information required for action analysis is stored in a set of *plan fragment tables*, one for each type of geom. Conceptually, a plan fragment table is a dispatch table, indexed by the invariants signature of the geom and the type of constraint to be satisfied. Each plan fragment in the table specifies how to move the geom to satisfy the new constraint using only available degrees of freedom, and specifies what new invariants the geom will have after the action is performed.

4.1.2. Geometric planning

Geometric planning begins by selecting a constraint which can be satisfied, and performing the appropriate measurements and actions. While the brick is initially free to move, it does have an arbitrary configuration C_0 in the numerical model, as shown in Fig. 3. The particular values of the brick's configuration variables do not affect the symbolic model.

A trace of GCE's solution process serves to illustrate the geometric planning. GCE decides to satisfy **dist:point-point**$(\boldsymbol{g}_1, \boldsymbol{b}_1, 0)$ first; it could have chosen any of the constraints. To satisfy this constraint, GCE measures the vector from $\boldsymbol{b}_1$ to $\boldsymbol{g}_1$. It then translates the brick by that vector, leaving the brick in configuration C_1, shown in gray. If **dist:point-point**$(\boldsymbol{g}_1, \boldsymbol{b}_1, 0)$ is to remain satisfied when future actions alter the brick's configuration, those future

[5]The semantics of the **dist:point-line** constraint allows the point to be a specified distance from the infinite line which is the extension of the line segment.

actions must be restricted to rotations about $\boldsymbol{g}_1$ (or equivalently, about $\boldsymbol{b}_1$). GCE ensures this by marking point $\boldsymbol{b}_1$ on the brick as being an invariant point.

GCE generates this sequence of measurements and actions by looking up the appropriate template in the plan fragment table, and binding the template's variables appropriately. Initially the brick has no invariants (i.e., its invariants signature is IR[000_00_0_000]). The plan fragment that is invoked contains the following information (descriptions in this and subsequent figures have been syntactically "cleaned up" for ease of reading):

```
Geom type: rigid-body
Constraint: dist:point-point(?geom1, ?geom2, ?d)
Invariants signature: IR[000_00_0_000]

Measurements and actions:
begin
?fixed = fixed-geom(?geom1, ?geom2);
?free = free-geom(?geom1, ?geom2);
?sphere = make-sphere(?fixed, ?d);
?dest = projection(?free, ?sphere);
?parent = top-level-parent(?free);
translate(?parent, vec-diff(global-loc(?dest),
                            global-loc(?free)));
end;

Bookkeeping:
if ?d == 0
  then add-invariant-point(?free, ?parent)
  else add-2D-constrained-point(?free, ?sphere, ?parent);
```

Explanation:
Geom *?parent* is free to translate. A **dist:point-point** constraint must be satisfied between point *?fixed*, whose global position is known to be invariant, and point *?free* on *?parent*. Therefore *?parent* is translated by the vector from the current global position of *?free* to a point on the sphere of radius *?d* around point *?fixed* with known global position. This action removes one translational degree of freedom if *?d* is non-zero, and removes all three translational degrees of freedom if *?d* is zero.

The plan fragment specifies how to move geom of type **rigid-body**, with an invariants signature of IR[000_00_0_000], to satisfy the constraint. The fixed and free geoms—both of which are points—are determined via functions called **fixed-geom** and **free-geom**, which check the invariant statuses of

?geom1 and *?geom2*. The effect is to assign a directionality to the constraint. In this example, $\boldsymbol{g}_1$ is the fixed geom and $\boldsymbol{b}_1$ is the free one. Since $\boldsymbol{b}_1$ is embedded in a rigid-body, the rigid body plan fragment table is used, and all operations (e.g., **translate**) are applied to the parent rigid body. The function **top-level-parent** follows the **parent** relation transitively until it reaches a geom which has no parent. The function **global-loc** returns the location (position for a point, orientation for a vector) of a primitive geom in the global coordinate system. The **projection** function is used to calculate the minimum alteration to the brick's current position that will satisfy the constraint. The textual explanation—with variable names replaced by their bindings—helps a user of GCE to understand the solution process.

After moving the brick, the plan fragment updates the invariants record of the brick to show that it has one invariant point, since the distance parameter of the constraint was zero. Note that the bookkeeping section of the plan fragment is responsible for noticing that a point is a degenerate case of a sphere (i.e., a sphere of radius zero). The invariants record of the brick now has a signature of IR[100_00_0_000].

GCE next chooses to satisfy **dist:point-point**$(\boldsymbol{g}_3, \boldsymbol{b}_3, 0)$; again, either of the remaining constraints could have been chosen. GCE measures the vector $\boldsymbol{v}_1$ from $\boldsymbol{g}_1$ to $\boldsymbol{g}_3$ (where $\boldsymbol{b}_3$ must be placed), and vector $\boldsymbol{v}_2$ from $\boldsymbol{g}_1$ to $\boldsymbol{b}_3$ (shown in its new location as $\boldsymbol{b}'_3$ in Fig. 3). These two vectors are shown as dashed lines in Fig. 3. Since the desired distance between the two points is zero, the problem can be solved only if the point $\boldsymbol{g}_3$ lies on a sphere centered at $\boldsymbol{g}_1$, with radius $|\boldsymbol{v}_2|$.

In order to move the brick, GCE requires a line about which to rotate it. The point $\boldsymbol{b}_1$ lies on this line, and if the rotation is to move $\boldsymbol{b}_3$ to coincide with $\boldsymbol{g}_3$, one acceptable line direction is the normal to the plane in which $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ lie, i.e., $\boldsymbol{v}_1 \times \boldsymbol{v}_2$ The amount to rotate the brick is the angle between these vectors, measured from $\boldsymbol{v}_1$ to $\boldsymbol{v}_2$. Therefore, GCE rotates the brick about $\boldsymbol{b}_1$ around vector $\boldsymbol{v}_1 \times \boldsymbol{v}_2$ by the angle between $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$. This action brings the brick to configuration $\boldsymbol{C}_2$, which satisfies **dist:point-point**$(\boldsymbol{g}_3, \boldsymbol{b}_3, 0)$ without violating **dist:point-point**$(\boldsymbol{g}_1, \boldsymbol{b}_1, 0)$. This action also removes two of the remaining rotational degrees of freedom; in order to preserve the two already-satisfied constraints, all future actions must be rotations about line segment $\overline{\boldsymbol{g}_1\boldsymbol{g}_3}$.

Once again, the sequence of measurements and actions is obtained by direct lookup in the plan fragment table. The actual measurements and actions are more complicated than described above, in order to handle the general case of a non-zero distance:

Geom type: **rigid-body**
Constraint: **dist:point-point**(*?geom1, ?geom2, d*)
Invariants signature: IR[100_00_0_000]

Measurements and actions:

```
begin
?fixed = fixed-geom(?geom1, ?geom2);
?free = free-geom(?geom1, ?geom2);
?parent = top-level-parent(?free);
?point = get-invariant(?parent, "Invariant points", 1);
?v1 = vec-diff(global-loc(?fixed), global-loc(?point));
?v2 = vec-diff(global-loc(?free), global-loc(?point));
?sphere1 = make-sphere(?fixed, ?d);
?sphere2 = make-sphere(?point, mag(?v2));
?circle = intersect(?sphere1, ?sphere2);
if ?circle == null
  begin
    if (mag(?v1) + mag(?v2) < ?d)
      then ?error = ?d - (mag(?v1) + mag(?v2))
      else ?error = abs(mag(?v1) - mag(?v2)) - ?d;
    error("Dimensionally inconsistent", ?error)
  end
?dest = projection(?free, ?circle);
?v3 = vec-diff(global-loc(?dest), global-loc(?point));
?cp = cross-prod(?v2, ?v3)
rotate(?geom, global-loc(?point), ?cp, vec-angle(?v2, ?v3, ?cp));
end;
```

Bookkeeping:

```
if ?d == 0
  add-invariant-point(?free, ?parent)
  else add-1D-constrained-point(?free, ?circle, ?parent);
```

Explanation:
Geom *?parent* has zero translational degrees of freedom, but may rotate about *?point*. If the points *?fixed* and *?free* have distances from *?point* which differ by no more than *?d*, the problem is solved by rotation about *?point*. Otherwise, the problem is dimensionally inconsistent. If *?d* is zero, geom *?parent* is left with one degree of freedom; otherwise it has two degrees of freedom.

A new feature of this plan fragment is the use of conditional statements to check the values of quantities. The two spheres *?sphere1* and *?sphere2* will not intersect in the following situations:

$$|?v1| - |?v2| > ?d,$$

$$|?v2| - |?v1| > ?d,$$

$$|?v1| + |?v2| < ?d.$$

In these situations, *?circle* will be null. An error value is calculated to indicate the severity of the problem. In all other cases, a solution is possible.[6] Since, in the brick example, *?d* is zero, another invariant point is added, and the invariants signature becomes IR[200_00_0_000].

To satisfy the final constraint, **dist:point-point**$(g_2, b_2, 0)$, GCE constructs a perpendicular from b_2 to $\overline{g_1 g_3}$, and creates a circle with radius equal to the magnitude of the perpendicular, center equal to the base of the perpendicular, and axis equal to the direction of line segment $\overline{g_1 g_3}$. If the circle is non-degenerate (i.e., has a non-zero radius), and it intersects point g_2, a solution is obtained by rotation about line segment $\overline{g_1 g_3}$. This action brings the brick to configuration C_3, which satisfies all three **dist:point-point** constraints. If the circle is degenerate (i.e., a point), no actions are taken, and no degrees of freedom are constrained. In the non-degenerate case, the action reduces the brick's remaining degrees of freedom to zero, by adding another invariant point.

4.1.3. The canonical nature of action analysis

Action analysis provides a simple way of decoupling the constraints pertaining to a single geom. It may be understood in the context of rewriting systems. A set of rewrite rules is *canonical* when all the normal forms of each expression are identical [3]. In such cases, the order in which the rules are applied does not matter; the result is always the same. When a set of rules is canonical, any applicable rule may be invoked, and "progress" will be made toward the solution. No ordering of the rules need be done, although it may be useful to guide the order of rule invocations to improve the efficiency of the process. Similarly, action analysis may be viewed as the process of repeatedly updating a geom's invariants record. Action analysis is canonical in the sense that, regardless of the order in which the constraints are satisfied, the invariants record of the geom at the end of the process is always the same.

Action analysis is shown to be canonical in the domain of rigid-body kinematics in [10, pp. 80–81, 247–249]. A proof has not yet been attempted in the expanded geometric domain of GCE, but it seems to be a natural extension of the existing proof.

4.2. Example 2: constraints on a circle

The brick problem illustrated how action analysis is used to generate a sequence of measurements and actions to satisfy a set of geometric constraints.

[6] If the vectors stored in registers *?v2* and *?v3* are gratuitously coincident, the cross-product vector stored in *?cp* will have a magnitude of zero. In this situation, the **rotate** operator performs no action.

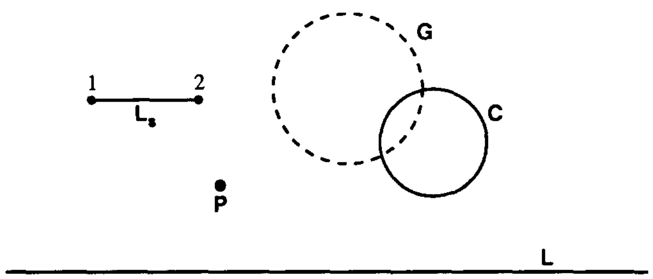

Fig. 4. Constraining a circle—initial conditions.

For each of these constraints, one geom must be invariant. This condition, however, is not always encountered in GCSPs. Often, geoms interact with each other in more complex ways that require the satisfaction of constraints between partially constrained geoms. This corresponds to the solution of nonlinear simultaneous equations in the algebraic domain.

A problem involving constraints that can only be solved by considering their interactions is shown in Fig. 4. This problem is a planar problem, i.e., all geoms are constrained to lie in a particular plane. The problem involves the following geoms:

- A circle C, of fixed position, orientation, and radius (i.e., grounded).
- An infinite line L, of fixed position and orientation (i.e., grounded).
- A grounded point P.
- A line segment L_S, of fixed length, free to translate and rotate within the same plane as C and L. The invariants of the line segment record that one endpoint is constrained to a two-dimensional locus (a plane), and the line segment's vector is constrained to a one-dimensional locus (perpendicular to the plane's normal); the invariants signature is IR[001_01_1_000].
- A circle G, free to translate in the same plane as C and L, as well as free to change radius; however, the axis of the circle is constrained to be the same as the normal to the plane: IR[001_10_0_000]. G is shown as a dashed circle in Fig. 4.

The additional constraints to be solved are:

(1) **dist:point-point**(**end-1**(L_S), P, 0),
(2) **dist:point-point**(**end-2**(L_S), **center**(G), 0),
(3) **dist:line-circle**(L, G, 0),
(4) **dist:circle-circle**(C, G, 0).

These constraints will be referred to by number in the following discussion. Since the constraints can be satisfied in any order, they will "arbitrarily" be attempted in the order in which they appear above.

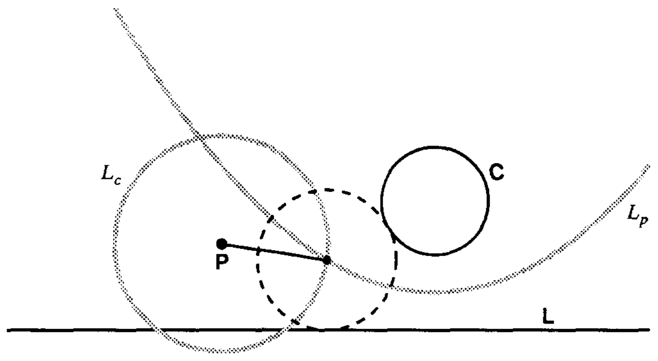

Fig. 5. Constraining a circle—solution.

4.2.1. Geometric planning: action analysis

Action analysis can be used to satisfy most of these constraints. Constraint (1) can be satisfied because point $\boldsymbol{P}$ is grounded. Therefore, line segment $\boldsymbol{L}_{\mathrm{S}}$ is translated to bring **end-1**$(\boldsymbol{L}_{\mathrm{S}})$ into coincidence with point $\boldsymbol{P}$. Constraint (2) cannot yet be satisfied, because neither the center of circle $\boldsymbol{G}$ nor **end-2**$(\boldsymbol{L}_{\mathrm{S}})$ are grounded.

Constraint (3) can be satisfied by action analysis because line $\boldsymbol{L}$ is fixed. No restrictions can be placed on the location of the center of the $\boldsymbol{G}$, nor on its radius. The invariant that is added to the invariants record is of the "fixed-distance lines" class. This invariant records the distance from the circle perimeter to the line (in this case zero). It serves to indicate that, were the radius of the circle fixed, the center would be restricted to a one-dimensional locus, or, were the center fixed, the radius would be known. This relationship restricts one degree of freedom.

Constraint (4) can be satisfied because circle $\boldsymbol{C}$ is fixed. The combination of constraint (4) and the fixed-distance line invariant is used to deduce that the center of $\boldsymbol{G}$ is in fact restricted to a one-dimensional locus; this is the parabolic locus L_{p} shown in Fig. 5. The center of $\boldsymbol{G}$, which was previously constrained to a two-dimensional locus (the plane), is "promoted" to a one-dimensional locus.

Still, constraint (2) cannot be satisfied, since neither **center**$(\boldsymbol{G})$ nor **end-2**$(\boldsymbol{L}_{\mathrm{S}})$ have become grounded through the solution of other constraints. However, there is enough information to satisfy this constraint.

4.2.2. Locus analysis

Locus analysis determines where in global space certain classes of partially constrained geoms must lie. If a subgeom is embedded in a parent geom that is not yet grounded but which has some geometric invariants, that subgeom is restricted to lie in a subregion of space. The locus of possible locations for the subgeom is a function only of the subgeom's position within its parent geom, and of the parent geom's degrees of freedom. When two partially constrained geoms are related by a constraint, the globally acceptable locations for those geoms often may be derived by intersection of their locally determined loci of possible locations. Once the global location is known, action analysis is once again used to move the relevant geoms to satisfy the constraint.

A collection of *locus tables* describes the loci of points, lines, and vectors embedded in a geom as a function of the invariants of that geom. A *locus intersection table* enables deduction of globally acceptable locations for pairs of geoms constrained by multiple loci. If the intersection yields a finite set of points, the locus intersection table also contains information about the maximum number of real roots the intersection equation may have; a *branch variable* is introduced into the solution to let a user of degrees of freedom analysis specify which branch of the solution should be used for the problem solution.

Even though an intersection may have several branches (or solutions), the solutions are topologically equivalent in that all loci resulting from the intersection are of the same dimensionality.[7] Thus, a locus intersection is a single abstract solution which can be instantiated by choosing a branch variable value. In this manner, a class of instantiable solutions are represented by a vector of branch variables associated with a metaphorical assembly plan, and a specific solution by a vector of branch variable values.

4.2.3. Geometric planning: locus analysis

At the current stage of the solution, $\boldsymbol{L}_s$ has an invariant endpoint, a vector constrained to be normal to the plan of the problem, and a fixed dimension. Thus, $\boldsymbol{L}_s$ has one degree of freedom (a line segment has six DOFs in three space; an invariant point subtracts three DOFs, a 1D-constrained vector removes one, and an invariant dimension subtracts one, leaving one remaining DOF). Therefore, any points on the line segment must have no more than one DOF. The locus tables indicate that **end-2**$(\boldsymbol{L}_s)$ is restricted to a circle locus, shown as L_c in Fig. 5.

The location of **center**$(\boldsymbol{G})$ has already been restricted to the parabola locus L_p via the solution of constraints (3) and (4). This allows constraint (2) to be satisfied as follows:

(1) Intersect loci L_p and L_c. Since multiple intersections are possible, a branch variable is assigned to the chosen solution so that the same intersection may be chosen in a subsequent solution of the constraint system.

[7]For more general non-analytic or piecewise-analytic curves, such as splines, this may not be the case, thereby making locus analysis more complicated.

(2) Use action analysis to rotate L_S so that **end-2**(L_S) is coincident with the intersection point. This action grounds L_S.
(3) Use action analysis to translate circle G so **center**(G) is coincident with the intersection point. Using the information stored in the "fixed-distance lines" slot of the invariants record, set the circle's radius so the perimeter touches line L. These actions ground G.

Locus intersection, followed by another round of action analysis thus grounds the remaining geoms and completes the solution of this constraint problem.

4.3. Structure of the plan fragment tables

The plan fragments are small programs that satisfy a constraint without violating any of the invariants already pertaining to a geom. When a new geom type or constraint type expands the ontology of the system, new plan fragments must be written, and the plan fragment tables expanded. Each geom type has its own plan fragment table; thus, the plan fragment table for a circle is different from that of a line segment. Since the plan fragment table is accessed by the invariants signature of a geom, the number of entries in the plan fragment table depends on the number of possible invariants signatures for that geom.

The geom representations sometimes allow redundant descriptions of the same state. For example, a grounded line segment could be described by an invariant dimension, an invariant point, and an invariant vector (IR[001_01_1_000]). However, it could be described equally well by two invariant points (IR[002_00_0_000]). Thus, for each geometrically distinct state, a set of invariants records may describe the geom, forming an equivalence class. To minimize the number of plan fragments in each table, one member of each equivalence class is (arbitrarily) designated the "canonical" invariants record. Then, each plan fragment is written so that only canonical invariants records can result from satisfying the constraints. At present, this task is performed manually; automating this process, or at least checking it for consistency, would greatly improve the knowledge engineering process.

Many entries in the various plan fragment tables share a similar structure. For instance, moving a line segment with no invariants to satisfy a **dist:point-point** constraint uses the same measurements and actions as moving an unconstrained circle or rigid body. To re-use generic strategies, the plan fragments are written in MATHCODE, a Mathematica-based system for translating high-level code descriptions into lower-level languages [8]. A single MATHCODE routine can then be used in several different plan fragments.

Verification of the plan fragments is achieved by exhaustive unit testing which takes into account all possible geometric degeneracies. A "geometric

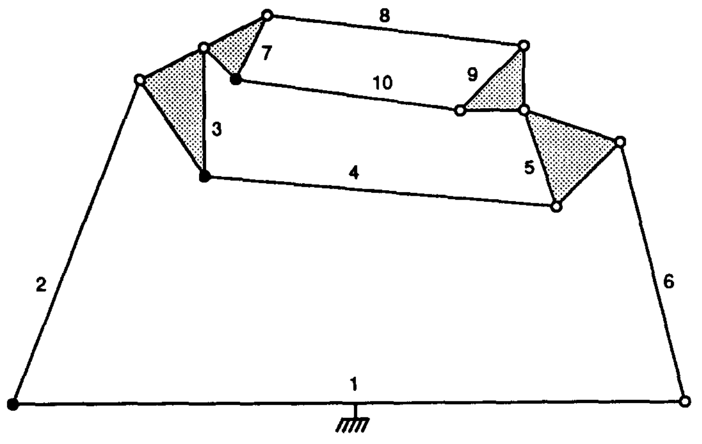

Fig. 6. A ten-bar linkage "recursively" composed of four-bar linkages.

construction checker", analogous to a theorem checker, would improve the verification process.

5. Loop and chain analysis

The previous examples were relatively simple in that all constraints could be solved with action and locus analysis being repeatedly used to "grow" the set of grounded geoms, thus allowing more constraints to be satisfied. More complex problems require solving subnetworks (e.g., loops or chains) of the constraint network in isolation, then reformulating those substructures as rigid bodies which can then be moved to solve other portions of the constraint network.

5.1. Example 3: hierarchical grouping of geoms

An example of a constraint system requiring analysis of constraint loops and chains is the ten-bar mechanical linkage shown in Fig. 6. Its structure is that of a four-bar linkage, whose coupler bar is composed of another four-bar, whose coupler is composed of yet another four-bar.

In Fig. 6, the geoms (called links in the mechanisms domain) have been labeled 1 through 10. All links are constrained to be in the plane. The joints connecting the links are modeled with **dist:point-point** constraints, all with zero distances. This system has three internal degrees of freedom, and hence requires additional constraints to fully constrain the system. The three joints in Fig. 6 which are solid black (connecting links 1 and 2, links 3 and 4, and links 7 and 10) are additionally constrained by **angle:vec-vec** constraints. In addition, link 1 is grounded (as indicated by the "foot" in the center of the link).

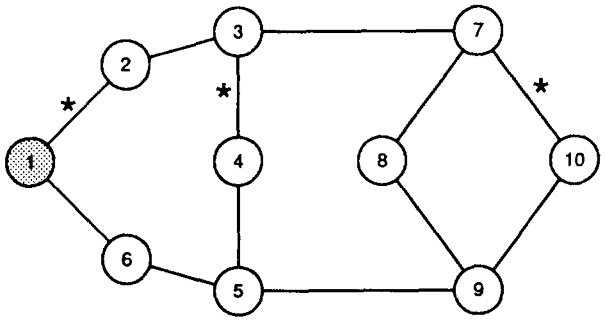

Fig. 7. Graph of the ten-bar linkage of Fig. 6.

In order to search for rigid substructures, degrees of freedom analysis employs a graph representation of the constraint system. In the constraint graph, nodes represent geoms, and arcs represent collections of one or more constraints (in subsequent discussion, the terms *geom* and *node* will be used interchangeably). Figure 7 shows the graph of the constraint system of Fig. 6 before solving. The node numbers correspond to the link numbering in Fig. 6. The grounded geom in this and subsequent graphs is shaded for easy identification.

In the absence of the constraints to be satisfied, each rigid-body geom in the system has three DOFs, since each body is constrained to the plane. The arcs in the graph of Fig. 7 which are marked with an asterisk restrict three DOFs, since they have **dist:point-point** constraints with zero distance, and angle constraints. Thus, satisfaction of the constraints on one of these arcs will cause the two geoms which they relate to be fixed rigidly with respect to each other. Acyclic collections of such geoms are called *chains*. Degrees of freedom analysis satisfies these constraints first, and reformulates each pair of geoms as a single rigid-body geom, also called a *macro-geom*. The resultant graph is shown in Fig. 8, where geoms 1 and 2 have been assembled to form geom 11, 3 and 4 have formed geom 12, and 7 and 10 have formed geom 13.

In the new graph, all remaining arcs have a single **dist:point-point** constraint that, in the plane, restricts two DOFs. No rigid chains remain, so degrees of freedom analysis next looks for rigid loops in the constraint graph. Consider what would happen if the loop of nodes 11, 12, 6, and 5 were to be satisfied using action and locus analysis. Each of the three non-grounded geoms has three DOFs, for a total of nine DOFs. The three constraints restrict only six DOFs, leaving three remaining DOFs. In other words, that loop would not be rigidly fixed. In contrast, consider loop 8–9–13–8. Were one of the geoms grounded, this loop would have zero DOFs. Finding a loop's degrees of freedom is analogous to determining the mobility of a mechanism [6], and the algorithms are quite similar.

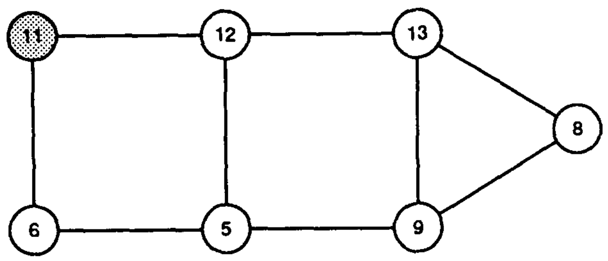

Fig. 8. Graph of Fig. 7 after replacing pairs of geoms constrained by driving inputs with macro-geoms.

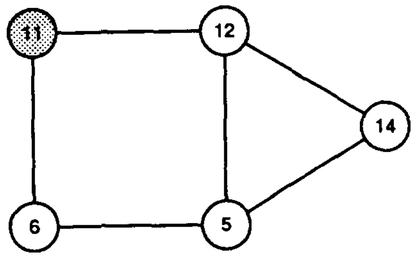

Fig. 9. Graph of Fig. 8 after assembling loop 8–9–13–8 and replacing it with the single node 14.

Degrees of freedom analysis identifies the loop with the lowest number of degrees of freedom, in this case, loop 8–9–13–8. It then temporarily grounds one of the geoms in this loop and uses action and locus analysis to solve for the constraints on the arcs connecting the three nodes. Next, it reformulates the composite geometry as a macro-geom, shown as node 14 in Fig. 9. This will in turn allow loop 12–14–5–12 to be reformulated as a macro-geom, which will enable the solution of the remaining constraints.

5.2. *Position analysis*

Position analysis is the term for the top-level strategy employed in degrees of freedom analysis. First, rigid chains are identified, solved, and reformulated as macro-geoms. Next, the loop with the fewest DOFs is identified, solved, and rewritten as a macro-geom. The process is repeated until the entire constraint graph is rewritten as a single node. Appendix A.3 describes the algorithm in detail.

5.2.1. *Underconstrained systems and iterative solutions*

Cases where no loop in the constraint system is rigid indicate one of two possible situations:

(a) the system is underconstrained, or
(b) the system has no closed-form solution.

In such cases, degrees of freedom analysis proceeds by identifying the loop with the fewest degrees of freedom, and adding as many redundant constraints to the system as are required to make the loop rigid. These redundant constraints are called *defaults.*

In case (a), the defaults serve to parameterize the remaining degrees of freedom in the system. In case (b), the constraints yield a near-minimal set of redundant generalized coordinates for use in an efficient iterative solution to the constraint system. Iterative solutions are formulated in the following manner: A set of parameter values are chosen for the defaults, and assembly of the geometry is attempted. If the geometry cannot be assembled, the **error** functions in the plan fragments accumulate an error proportional to the degree to which the assembly is incorrect. Traditional optimization schemes are then used to vary the default parameters until the error term vanishes.

Once degrees of freedom analysis has committed to solving a particular loop, it will not backtrack. Therefore, if the loop is degenerate in some way, a redundant generalized coordinate may be introduced when in fact solving a different loop first would have obviated the need for a redundant generalized coordinate. While this does not affect the quality of the answer to the GCSP, it does affect the efficiency of the solution process. The group-theoretic approach to finding degrees of freedom proposed by Hervé [7] may be useful in detecting degeneracies before loop solution, as evidenced by similar work by Popplestone et al. [17].

A complete discussion of loop and chain analysis would exceed the space limitations of this article. Extended discussion is found in [10].

5.2.2. Plan compilation issues

The TLA system was employed as a simulation "compiler" by storing the sequence of measurements and actions in a re-executable plan structure [9]. This allowed typically linear behavior, with a worst-case $O(n \log n)$, in the simulation of mechanisms, where n is the number of links. In its expanded scope, degrees of freedom analysis still utilizes a plan representation; however, rather than a linear array, a tree is used to store the plan. Each node in the plan tree has different exit points depending on the number of degrees of freedom absorbed by the constraint. Upon re-execution with different dimensions or constraint parameters, a new geometric degeneracy may arise, causing a new branch of the plan tree to be generated and stored. This allows caching solutions from various degenerate geometries. Currently, the tree-style plan representation is in prototype form.

6. Empirical and theoretical analysis

Degrees of freedom analysis provides low-order polynomial-time algorithms for the solution of GCSPs. Problems are solved in $O(cg)$ time, where c is the number of constraints, and g is the number of geoms. Details of the complexity analysis are provided in Appendix A.

6.1. Canonicality

In degrees of freedom analysis, the constraint satisfaction process is canonical. At any stage of the solution process, a number of choices may exist as to which constraint to satisfy next. Any choice may be made, with no affect on the final answer. Proving that the position analysis algorithm is canonical is done by proving that chain and loop rewriting are *confluent*, and then showing that this implies that the algorithm is canonical.

Borrowing the terminology used in [3], which applies to expression rewriting, confluence is defined in this context as the property that whenever a subgraph S in the constraint graph can be rewritten in two different ways, say to I_1 and I_2, then I_1 and I_2 can both be rewritten to some common graph S'. A proof of the canonical nature of position analysis is found in [10, pp. 140–145].

6.2. Empirical comparisons

Degrees of freedom analysis was empirically validated in the domain of kinematics with an implemented computer program called The Linkage Assistant (TLA). This program has performed kinematic simulation of complex mechanisms in a more computationally efficient manner than other existing programs. Efficiency increases of two orders of magnitude were observed on medium-sized examples involving on the order of a hundred constraints, when compared with ADAMS, a mechanism simulator using a maximally redundant generalized coordinate representation and iterative numerical solution [1].

The graph of Fig. 10 shows the timing analyses of ADAMS and TLA as a function of the number of bodies in a mechanism. The dashed line shows the time per iteration for ADAMS; it is a polynomial curve proportional to $n^{2.17}$, where n is the number of links in the mechanism. This indicates the efficiency of the sparse matrix routines employed by ADAMS. Typically, between 2 and 12 iterations are required to solve a single step of the simulation, as indicated by the gray area. In contrast, the behavior of TLA (re-using its compiled plan) is linear, and is substantially more efficient.

7. Conclusion

While symbolic solution of the algebraic equations describing geometric constraints is NP-complete in general, degrees of freedom analysis allows generating closed-form solutions, or efficient iterative solutions, to GCSPs

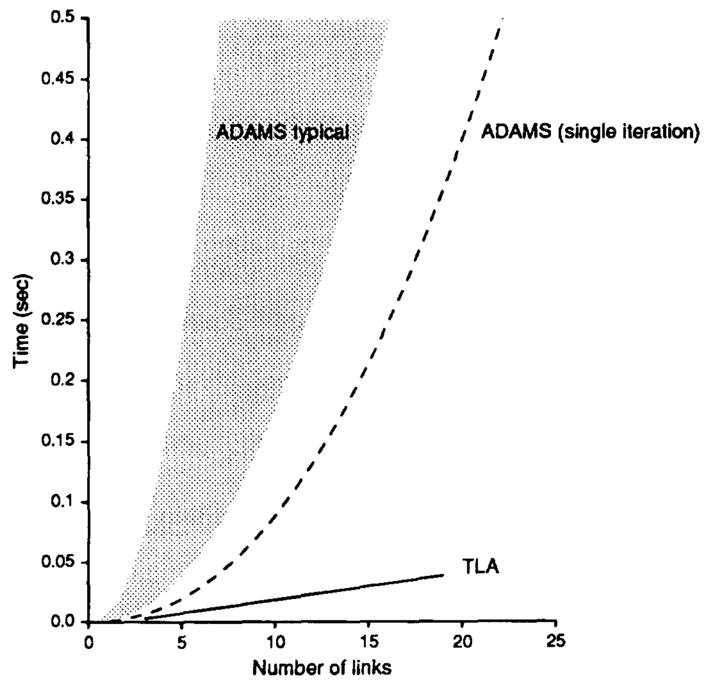

Fig. 10. Comparison of runtimes of TLA and ADAMS.

in polynomial time. This avoids the problem of computational intractability found in symbolic algebra systems. Typically, the resulting closed-form solutions may be executed in time linearly proportional to the size of the constraint system being analyzed, and are substantially more efficient than iterative numerical techniques.

The power in degrees of freedom analysis comes from the use of a metaphor of incremental assembly. This allows for maximal decoupling of the constraints. Similar operational semantics can be found in the work of Arbab and Wang [2], Pabon et al. [14], and Wilk [20]. However, in these systems, methods must be provided to solve a new constraint in the context of the object plus *all* the constraints currently satisfied. The number of constraint satisfaction methods can grow quickly with the types of objects and types of constraints—$O(c^n)$, where c is the number of constraint types, and n is the number of DOFs in an unconstrained geom. By using DOFs as equivalence classes, degrees of freedom analysis coalesces many of these states, thereby creating a more manageable search space.

The work of Arbab and Wang [2] and Pabon et al. [14] used iterative numerical techniques whenever a constraint loop was found. In contrast, locus analysis allows solving many such problems in closed form. Locus analysis can be considered a generalization of the "cycle finder" described by Popplestone in [16].

There are a number of interesting topics for further extending degrees of freedom analysis. Currently, the plan tree is recomputed whenever the topology of the constraint problem changes. This can be inefficient if constraints are only being added. However, re-use of the tree may result in the retention of previously-added redundant generalized coordinates which are no longer needed.

Experiments have begun in mixing iterative and closed-form solutions within the plan fragments themselves. For example, if a point geom is confined to a one-dimensional locus described by a sixth-degree polynomial, and must be positioned a certain distance from another point, a locally iterative solution can be used to compute the intersection of the locus and a sphere. This is more efficient than employing a general iterative solver in an "outer loop" invoking the entire assembly plan.

7.1. Relation to general constraint satisfaction

Mackworth's description of constraint graphs, and the concepts of node, arc, and path consistency, were formulated for predicates with finite, discrete-valued variable domains [12]. In GCSPs, the variables are continuous, with infinite domains. Node, arc, and path consistency are more difficult in such situations.[8]

In the context of the graph representation of general constraint systems, degrees of freedom analysis may be thought of as a layered solution to the GCSP. In the original formulation of the problem, the nodes in the constraint graph represent real values for the configuration variables of the geoms, and the arcs represent constraints. A "meta-system" may be devised in which the graph nodes represent the number of degrees of freedom of the system, and the arcs represent constraints. The meta-system maps the continuous variables into the discrete-valued DOF space.

In the meta-system, node, arc, and path consistency are used to remove degrees of freedom from the nodes (since any degrees of freedom on the nodes are incompatible with constraints on adjacent arcs). Node consistency corresponds to grounding a geom, arc consistency corresponds to solving a chain, and path consistency corresponds to solving loops. Every time a constraint is removed in the meta-system, a set of measurements and actions is posted as a side-effect. These side-effects solve the GCSP as originally formulated.

From this analogy, it is seen that there are some broad concepts that can be re-used in formulating constraint satisfaction problems for other domains. The notion of abstracting some continuous space (e.g., position,

[8]Interval arithmetic, while technically continuous, discretizes the domain by considering only a finite set of interval endpoints.

orientation, dimension) into a discrete space (e.g., degrees of freedom) may apply to other domains. Designing algorithms that make use of monotonic trends (such as the reduction of degrees of freedom of a geom) tend to lead to polynomial-time algorithms. Planning metaphors can help to guide search. Creative representation shifts will be required to use these principles in other domains, but if they can be found, the benefits may be substantial.

Appendix A. Algorithms

The section describes, at a high level, the algorithms for the major functional components of GCE. Algorithms for action analysis and locus analysis are not given; they are similar to those found in [10].

A.1. Solving chains

Satisfying the constraints on a rigid chain of arbitrary length proceeds recursively by satisfying the constraints between a pair of geoms, and rewriting the pair of geoms as a single macro-geom. The algorithm for identifying a pair of geoms which can be rewritten as a macro-geom, and solving the appropriate constraints is:

Algorithm SOLVE-RIGID-CHAINS. *Algorithm for recognizing and rewriting topologically rigid chains in a constraint graph as macro-geoms.*

Input: Constraint graph G.
Other variables: I for temporary storage of a geom's invariants record.

```
     procedure SOLVE-RIGID-CHAINS(G):
       begin
1        for arc a in the constraint system do
2          if a's constraints imply a rigid connection between the
             geoms (see [10])
3          then
             begin
4              $g_1 \leftarrow$ one node connected to a
5              $g_2 \leftarrow$ the other node connected to a
6              if grounded($g_2$)
7              then swap($g_1, g_2$)
```

Comment: *At this point, the following encoding has been established: If one of the geoms is grounded, it is stored in* g_1.

```
8              if not grounded($g_1$)
```

```
 9              then
                  begin
10                  $I \leftarrow$ copy-invariants-record($g_1$);
11                  ground!($g_1$);
                  end
12              ACTION-ANALYSIS(constraints in $a$);
13              Replace $g_1, a, g_2$ with macro-geom $m$;
14              set-invariants-record($m, I$);
              end
        end
```

Using this algorithm, all rigid chains in a constraint system can be reformulated in O(a) time, where a is the number of arcs in the graph.

A.2. Solving loops

Depending on the geometric domain, there is a limit to the size of a loop which can be assembled into a rigid macro-geom. This limit is six geoms for general bodies in 3D space, and three geoms for bodies in 2D space.[9]

In the algorithm described below, loops are identified in *stages*. A stage is a list of sequences of node numbers that describe a path through the graph. Sequences in stage s each contain $s + 1$ nodes. A sequence describes a loop if the first and last numbers in that sequence are equal. To avoid identifying the same loop multiple times (e.g., 8–9–13–8 and 13–8–9–13), a canonical form is required in which the first node number in the loop is the smallest, and the second node number is less than the penultimate one:

Algorithm IDENTIFY-LOOPS. *Algorithm for identifying all loops of size l or less in a constraint graph.*

Input: G, the constraint graph
l, the maximum number of nodes in a loop.
Other variables: Constraint graph connectivity array C,
where $C[i]$ contains the list of nodes connected to node i.
Stage array (stage[s] contains the sequences in stage s).
Output: The loops found.

```
     procedure IDENTIFY LOOPS($G, l$):
       begin
1        $C \leftarrow$ make-connectivity-array($G$);
```

[9]Larger loops may be handled by degrees of freedom analysis; however, the absence of any rigid loops implies an iterative solution will be necessary. Defaults will be added to the system until one of the loops becomes rigid. In this process, rewriting the chains (formed using the defaults) as macro-geoms will reduce the loop size to the limitations specified here.

```
 2      for $i \in$ node numbers do
 3        for $j$ in $C[i]$ do
 4          if $i < j$
 5          then stage[1] $\leftarrow$ stage[1] + $i, j$
 6      for $s \leftarrow 2$ until $l$ do
 7        for $x_1, x_2, \ldots, x_s$ in stage[$s - 1$] do
 8          for $j$ in $C[x_s]$ do
 9            if $x_s \neq x_1$
10              and $x_1 \leqslant j$
11              and $\forall k,\ 1 < k \leqslant s.\ x_k \neq j$
12            then stage[$s$] $\leftarrow$ stage[$s$] + $x_1, x_2, \ldots, x_s, j$
13      loops $\leftarrow \emptyset$;
14      for $s \leftarrow 1$ until $l$ do
15        for $x_1, x_2, \ldots, x_{s+1}$ in stage[$s$] do
16          if $x_1 = x_{s+1}$ and $x_2 < x_s$
17          then loops $\leftarrow$ loops $+ x_1, x_2, \ldots, x_{s+1}$
18      return loops;
    end
```

Lines 9 through 11 provide the preconditions for the next node to be a valid continuation of the sequence: the first and last node numbers must not be equal (this would indicate a loop has already been found); the new node number must be greater than the first number; and, the new number must not already be a member of the sequence, unless it matches the first node number, forming a loop. Line 16 checks if a sequence is a canonical description of a loop. In the case where the number of arcs in the graph is comparable to the number of nodes (typical of many GCSPs), the complexity of this algorithm can be shown to be linear in the number of nodes [10].

The constraints in a loop are solved using the following approach: choose a ground node (if one does not exist), and then switch between action and locus analysis until constraints have been satisfied or no further inference is possible:

Algorithm SOLVE-LOOP. *Algorithm for solving the constraints pertaining to a loop.*

Input: Loop L.
Other variables: M a temporary transform matrix.
Output: RGC, a list of any redundant generalized coordinates used.

```
    procedure SOLVE-LOOP($L$):
      begin
 1      RGC $\leftarrow \emptyset$;
```

```
2     $g_1 \leftarrow$ grounded geom of$L$ (or $\emptyset$ if no geom is grounded);
3     $g_2 \leftarrow \emptyset$;
4     **if** $g_1 = \emptyset$
5     **then** $g_2 \leftarrow$ an acceptable ground geom
        (by the kinematic inversion decision procedure—see [10])
6     **else if** $g_1$ is an acceptable ground geom
7     **then** $g_2 \leftarrow g_1$
8     **else** $g_2 \leftarrow$ an acceptable ground geom
```

Comment: *At this point, the following encoding has been established: If g_1 is null, then no geom on the original loop was grounded. Otherwise, g_1 is the original grounded geom, and g_2 is the geom being used as ground for the loop solution.*

```
9     if $g_1 \neq \emptyset$ and $g_2 \neq g_1$
10    then $M \leftarrow$ transform($g_1$)
11    REPEAT: ACTION-ANALYSIS(all constraints in $L$);
12    LOCUS-ANALYSIS(all constraints in $L$)
13    if all constraints are satisfied
14    then
        begin
15        rewrite all geoms as macro-geom $G$;
16        if $g_1 \neq \emptyset$ and $g_2 \neq g_1$
17        then move $G$ by inverse(transform($g_2$))$\cdot M$;
18        return RGC;
        end
19    else if any constraints were satisfied in lines 11 or 12
20    then goto REPEAT
21    else
        begin
22        add a redundant constraint restricting one DOF;
23        add corresponding generalized coordinate
            (the real argument of the redundant constraint) to RGC;
24        goto REPEAT;
        end
   end
```

A.3. Top-level algorithm

The top-level algorithm finds the positions, orientations, and dimensions of all the geoms so that all constraints are satisfied. First, any unary constraints are trivially satisfied; in GCE, these are constraints on the dimension of one geom, so the dimension is adjusted and the dimensional DOF is fixed. Then, chains and loops are solved and rewritten, until the constraint graph has been reduced to a single node.

Algorithm POSITION-ANALYSIS. *Algorithm for finding a closed-form solution to a given GCSP, if one exists. Otherwise, the algorithm finds a solution with a minimal number of redundant generalized coordinates, which can then be used by an iterative numerical solver.*

Input: G, the constraint graph,
l, the maximum loop size.
Other variables: *loop*, a loop that is to be solved.
Output: RGC, a list of any redundant generalized coordinates used.

```
     procedure POSITION-ANALYSIS($G, C, l$):
       begin
 1       RGC ← ∅;
 2       Solve all dimensional (i.e., unary) constraints;
 3       REPEAT: SOLVE-RIGID-CHAINS($G$);
 4       if $G$ is a single node
 5       then return RGC;
 6       $L$ ← IDENTIFY-LOOPS($G, l$);
 7       if $L = \emptyset$
 8       then
           begin
 9           add a redundant constraint restricting one DOF;
10           add corresponding generalized coordinate
               (the real argument of the redundant constraint) to RGC;
11           goto REPEAT;
           end
12       for $l \in L$ do
13         CLASSIFY-LOOP($l$);
14       loop ← PICK-LOOP($L$);
15       RGC ← RGC + SOLVE-LOOP(loop);
16       goto REPEAT;
       end
```

The PICK-LOOP algorithm is responsible for choosing the best loop to solve, given the choices available. Assuming that SOLVE-RIGID-CHAINS is linear in the number of arcs, a, the complexity of POSITION-ANALYSIS is $O(na)$, where n is the number of nodes in the constraint graph. This results from the fact that, each time the loop in lines 3 through 16 is executed, the size of the constraint graph is decreased by at least one node.

Acknowledgement

The author would like to thank Phil Agre, Harry Barrow, Phil Husbands, and Robin Popplestone for comments and discussion. Mahesh Kanumury and Walid Keirouz are principal co-implementors of GCE, and contributed greatly to the ideas in this paper. Jahir Pabon and Robert Young provided useful comments on an earlier draft of this paper.

References

[1] *ADAMS Users Manual* (Mechanism Dynamics, Inc., Ann Arbor, MI, 1987).

[2] F. Arbab and B. Wang, A constraint-based design system based on operational transformation planning, in: J.S. Gero, ed., *Proceedings 4th International Conference on Applications of Artificial Intelligence* (Spinger, Berlin, 1989).

[3] A. Bundy, *The Computer Modelling of Mathematical Reasoning* (Academic Press, London, 1983).

[4] G. Celniker and D. Gossard, Deformable curve and surface finite-elements for free-form shape design, *Comput. Graph.* **25** (4) (1991) 257–266.

[5] R.H. Gallagher, *Finite Element Analysis: Fundamentals* (Prentice-Hall, Englewood Cliffs, NJ, 1975).

[6] R.S. Hartenberg and J. Denavit, *Kinematic Synthesis of Linkages* (McGraw-Hill, New York, 1964).

[7] J.M. Hervé, Analyse structurelle des mécanismes par groupes de déplacements, *Mech. Mach. Theor.* **13** (1978) 437–450.

[8] E. Kant, F. Daube, W. MacGregor and J. Wald, Scientific programming by automated synthesis, in: M.R. Lowry and R.D. McCartney, eds., *Automating Software Development* (AAAI Press, Menlo Park, CA, 1991) 169–205.

[9] G.A. Kramer, Solving geometric constraint systems, in: *Proceedings AAAI-90*, Boston, MA (1990) 708–714.

[10] G.A. Kramer, *Solving Geometric Constraint Systems: A Case Study in Kinematics* (MIT Press, Cambridge, MA, 1992).

[11] Y. Liu and R.J. Popplestone, Symmetry constraint inference in assembly planning: automatic assembly configuration specification, in: *Proceedings AAAI-90*, Boston, MA (1990) 1038–1044.

[12] A.K. Mackworth, Consistency in networks of relations, *Artif. Intell.* **8** (1) (1977) 99–118.

[13] A. Nevins, Plane geometry theorem proving using forward chaining, Artificial Intelligence Lab Tech. Rept. No. 303, MIT, Cambridge, MA (1974).

[14] J. Pabon, R. Young and W. Keirouz, Integrating parametric geometry, features, and variational modeling for conceptual design, *Int. J. Syst. Autom. Res. Appl. (SARA)* **2** (1992) 17–37.

[15] R.P. Paul, *Robot Manipulators: Mathematics, Programming, and Control* (MIT Press, Cambridge, MA, 1981).

[16] R.J. Popplestone, A language for specifying robot assembly, Department of Artificial Intelligence Res. Rept. No. 29, University of Edinburgh, Edinburgh, Scotland (1977).

[17] R.J. Popplestone, A.P. Ambler and I.M. Bellos, An interpreter for a language for describing assemblies, *Artif. Intell.* **14** (1) (1980) 79–107.

[18] W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes: The Art of Scientific Computing* (Cambridge University Press, Cambridge, England, 1986).

[19] J.R. Rossignac, Constraints in constructive solid geometry, in: *Proceedings 1986 Workshop on Interactive 3D Graphics* (1986) 93–110.

[20] M.R. Wilk, Equate: an object-oriented constraint solver, in: *Proceedings ACM Conference on Object-Oriented Programming Systems, Languages and Applications*, Phoenix, AZ (1991) 286–298.

[21] T.C. Woo and D. Dutta, Automatic disassembly and total ordering in three dimensions, *Trans. ASME, J. Eng. Indust.* **113** (2) (1991) 207–213.