# Geometric Constraint Solving: Algebraic Issues*

Christoph M. Hoffmann
Purdue University
West Lafayette, IN 47907

March 27, 1995

**Abstract**

Geometric constraint solvers naturally solve systems of nonlinear algebraic equations. When the geometric problem is well-constrained, the system of equations has a zero-dimensional set of solutions.

Graph-directed algebraic solvers analyze a constraint graph and deduce from this analysis a sequence of construction steps. Each construction step corresponds to solving a small, predetermined system of nonlinear equations. The complexity of the resulting algorithm is essentially quadratic in the number of geometric elements. Moreover, it is possible to systematically explore all solutions of the constraint problem.

In effect, such constraint algorithms solve systems of nonlinear algebraic equations by a decomposition process. The question is posed whether the decomposition can be abstracted and fashioned into efficient algorithms for solving nonlinear systems purely algebraically with comparable efficiency.

## 1 Introduction

Geometric constraint solvers are important components of CAD systems. They are used to support the following design paradigm:

> The user prepares a rough sketch composed from line segments and circular arcs, say, and annotates the sketch with constraints prescribing distances, angles, tangency, concentricity, etc. The constraint solver then computes a solution sketch that satisfies those constraints, if this is possible.

Figure **??** shows an example. The papers [1, 4, 3] explain in more detail how constraint solving is used in CAD and contain a literature survey. An on-line

*Work supported in part by ONR Contract N00014-90-J-1599, by NSF Grant CDA 92-23502, and by the NSF Purdue Engineering Research Center.


Figure 1: Left: Input to a geometric constraint solver. Not shown are tangency and perpendicularity constraints. Right: Output the solver.

multi-media *Electronic Primer On Geometric Constraint Solving* is also available on the world-wide web at

`http://www.cs.purdue.edu/homes/cmh/electrobook/intro.html`

Note that the primer includes instructions for running an example constraint solver.

Underlying the task of a geometric solver is to find a solution of a system of nonlinear algebraic equations. If the geometric problem is well-constrained, this system of equations defines a zero-dimensional ideal. When the expressive power of the constraint solver is suitably restricted, very efficient constraint solvers can be built. In particular, graph-directed algebraic solvers [9, 1] solve an interesting class of geometric problems and have a worst-case running time that is quadratic in the number of geometric elements. Moreover, those solvers can be structured such that the entire solution tree can be explored incrementally. Note, however, that the tree of solutions is in general exponential in the number of geometric elements. Consequently, exploring the entire set of solutions cannot be done within a quadratic time frame.

In this paper, we sketch very briefly how a graph-directed algebraic solver works. Then, the question is posed whether the techniques used by this class of solvers can be transposed into a purely algebraic setting. Doing so could benefit geometric constraint solvers, and it could create new techniques to solve polynomial systems efficiently.

## 2 Graph-Directed Algebraic Solvers

For the sake of conciseness and simplicity, the geometric constraint problem is considered in two dimensions. The three-dimensional problem is considerably harder; e.g., [8, 7]. The geometric elements are points, lines and circles with given radius. The constraints are distance, angle, radius, incidence, parallel, perpendicular, tangent, and concentric.

The input elements may have initial coordinates, but not all solvers need them to find a solution. In particular, graph-directed algebraic solvers do not

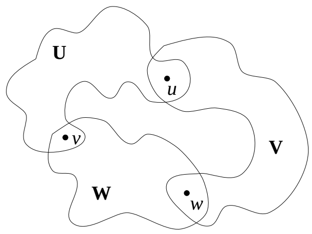


Figure 2: Schema for merging three clusters in the constraint graph. The shared geometric elements are represented by the vertices $u$, $v$ and $w$.

need them. Those solvers first construct a constraint graph whose vertices are the geometric elements and whose edges are the constraints between them. For example, a line segment of prescribed length has a triangle as constraint graph whose vertices are the two end points and the line on which the segment lies. The edges of the constraint graph are the two incidence constraints and the distance constraint.

The graph analysis is roughly as follows: Every graph edge and its two incident vertices are considered a *simple cluster*. Thereafter, triples of clusters are merged if they pairwise have a single vertex in common. However, the three clusters to be merged must not share a common vertex. See also Figure 2. If all clusters are eventually merged into a single cluster, then the geometric problem is *generically* solvable. Geometric problems that can be so solved are a subset of ruler-and-compass constructible configurations [9].

After the graph analysis has been done, the constructions determining the actual coordinates are carried out in the sequence of cluster merges. Each construction step solves a simple system of nonlinear equations, usually two quadratic ones.

A simple cluster corresponds to two geometric elements that can be placed relative to each other in a way that satisfies the constraints. As clusters are merged, more geometric elements are found that can be placed relative to each other obeying the constraints. Merging three clusters corresponds to placing three groups of geometric elements with respect to each other so that all constraints between them are satisfied. Some caveats are in order:

1. Sometimes, the position and orientation of the geometric figure with respect to a global coordinate system is not important. Thus, the final geometric figure is determined up to a translation and a rotation. The corresponding system of nonlinear equations is therefore not zero-dimensional but of dimension three.

2. The graph analysis determines generic solvability. There is no guarantee at this stage that a real solution can be so determined since the constraint values themselves are not taken into consideration.

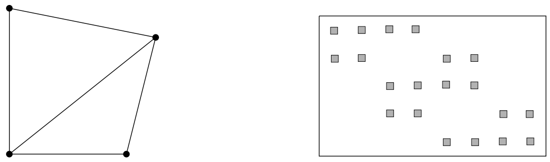

Figure 3: Left: A sequential constraint graph: The nodes may represent points, the edges distance constraints. Right: Structure of the associated equation system: Each matrix row represents an equation, the marks indicate variables occurrences.

3. The algorithm is somewhat more complex in that two parallel lines are not considered to form a simple cluster — the distance between them is undefined.

4. Three simple clusters whose vertices are all lines are not merged — one of the angle constraints between the lines is either contradictory or else redundant. If redundant, moreover, the three lines cannot be placed uniquely with respect to each other, so the problem is actually underconstrained.

Cluster merging is not unique. However, it can be proved that no matter how the merging sequence is determined, if one sequence obtains a single final cluster, then all sequences will. Furthermore, by using some very simple heuristics and an initial sketch, all merge sequences must produce congruent solutions [5].

# 3 Cluster Merging and Algebra

Assume that a merging sequence has been found in which, at each step, two simple clusters are merged with a third cluster. We call such a merge a *simple* merge. Then this sequence of simple merges corresponds to a sequential construction of the final geometric configuration that begins by placing two geometric elements and continues placing the remaining elements one-by-one. For example, the constraint problem shown in Figure 3 can be solved by a sequential construction.

Such sequential constructions correspond to an underlying equation system with a near-triangular structure: The value of three initial variables is chosen arbitrarily, corresponding to fixing a translation and rotation with respect to a global coordinate system. A fourth variable is found by solving a single equation. At that point, the first simple cluster has been constructed. Thereafter, the remaining variables are introduced in pairs. Every pair gives the coordinates of the next geometric element. Values are found by solving two simultaneous equations in two variables, usually quadratic equations.

We could say that such sequential problems correspond to block-triangular systems of equations, and that the graph analysis determines a permutation

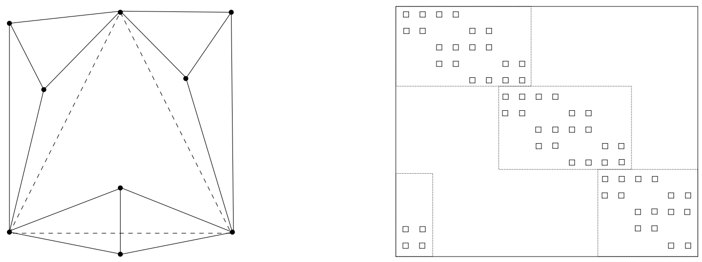

Figure 4: Left: Nonsequential constraint problem: Vertices represent points, solid edges explicit constraints, and dashed edges derived constraints. Right: Structure of the associated equation system.

of the variables that discovers this structure. Furthermore, every block has two additional variables and two additional equations in them. The fact that the equation degree is at most quadratic is less important, and is more of a computational convenience.

Things are more complex when constraint problems cannot be serialized. In such problems, three clusters will get merged at some point with at most one of them a simple cluster. The constraint solver does the associated construction very elegantly: Each of the three clusters to be merged consists of elements whose relative placement is known. The merge would be effected by a rigid-body motion of two of the clusters that matches the three shared geometric elements. By *measuring* in each cluster how two shared elements lie with respect to each other, *derived* constraints can be computed that reduce the computation to a sequential construction step placing one of the three shared geometric elements with respect to the other two. In the example in Figure 4 there are three derived constraints indicated by the dashed lines in the graph on the left.

Assume that this nonsimple merge happens only once, as the last merging step. Then the equation system can be partitioned into three separate subsystems, each with the block-triangular structure of a sequential construction problem, and each with three undetermined variables that can be chosen freely, resulting in nine degrees of freedom. The three subsystems interact with each other by sharing a pair of variables. Therefore, six of the nine degrees of freedom are constrained by the sharing, and the combined system has only three degrees of freedom. Note that this interacting structure can be compounded recursively by other nonsimple cluster merging

The challenge is to devise a way to capture the measuring of derived constraints by a simple symbolic computation. A precomputation of the relative position of two geometric elements as function of intra-cluster constraints is an unattractive approach, because the resulting polynomials quickly become unwieldy. For instance, restricting to points and distance constraints alone, the Cayley-Menger determinant can be used to derive the sixth distance constraint

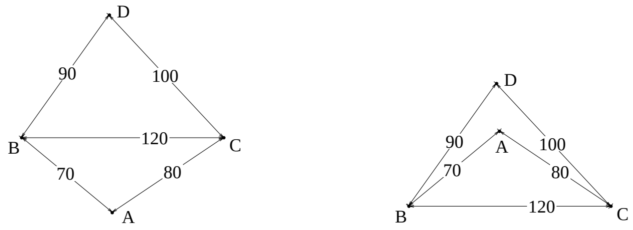


Figure 5: Different solutions of a constraint problem lead to different distance measurements between points A and D, and therefore also to different equations expressing the implied constraint.

between four points from the other five. The resulting symbolic expression is quite unattractive, arguing for the incorporation of this computation into a solver algorithm itself so the actual derivation of implied constraints can be delayed until the variables in question have values relative to each other.

The constraint derived depends on the type of interacting geometries the shared variables coordinatize. For example, when two points are involved, an equation of the form

$$(x_i - x_j)^2 + (y_i - y_j)^2 = d_{ij}^2$$

is derived where $d_{ij}$ is the measured distance between the points $(x_i, y_i)$ and $(x_j, y_j)$. If a point and a line are involved, on the other hand, the derived equations are

$$\begin{aligned} a_i x_j + b_i y_j + c_i &= d_{ij} \\ a_i^2 + b_i^2 &= 1 \end{aligned}$$

The derived equations do not constitute a general elimination of the other variables in the cluster. As example consider Figure 5. Depending on whether the solution on the left or on the right has been constructed, a different distance will be measured between points A and D. A new equation therefore is not necessarily in the ideal generated by the original system as it depends on a solution instance of the system. This leads to equations that have a lower degree than those derived by elimination. When constructing a different solution, as discussed in [1], those equations would change.

The constraint solver formulates such equations. It is not difficult to see that solving them is equivalent to doing a simple merge. More complicated cluster merging patterns can be formulated that require solving more complicated systems of algebraic equations as discussed in [1].

It would seem appropriate to seek ways in which the cluster merging can be formulated purely algebraically. This would mean in particular understanding the meaning of each variable, and knowing whether it is a point coordinate, a line coordinate, for this influences what equations to adjoin in order to capture the implied constraints.

One way would be to replicate variables that are shared between subsystems, and to formulate connecting equations that express the rigid-body motions required to align clusters upon merging. Since rotations are involved, trigonometric expressions would have to be expressed algebraically. Judging from preliminary work, this approach appears to be rather unattractive to us, unfortunately.

## 4 Conclusion

Graph-directed constraint solvers elegantly decompose the work of solving a large system of nonlinear equations. The decomposition rests on properties of the equation system and capitalizes on computations that are less general than the formal elimination used by symbolic computations such as Gröbner bases, resultants, or the Wu-Ritt method; e.g., [2, 10, 6]. Unfortunately, general elimination does not appear to produce competitive algorithms for solving the equation systems that are successfully solved by graph-directed constraint solvers.

The incomplete, instance-based elimination is the principal method by which such solvers achieve high speeds while retaining the flexibility of finding, if so directed, all solutions of the system. It would be nice to formalize this idea of deriving new constraints purely algebraically. Such a formalization should generalize the nature of the algebraic quantities involved so we would not have to know that the variables represent points or lines. This would lead to broader generalizations of the approach. Furthermore, the block sizes and the degrees of the equations capturing the overlap of subsystems should also be generalized.

## References


[1] W. Bouma, I. Fudos, C. Hoffmann, J. Cai, and R. Paige. A geometric constraint solver. *Computer Aided Design*, page to appear, 1995.

[2] B. Buchberger, G. Collins, and B. Kutzler. Algebraic methods for geometric reasoning. *Annual Reviews in Computer Science*, 3:85–120, 1988.

[3] X. Chen and C. Hoffmann. Design compilation for feature-based and constraint-based CAD. In *Proc 3rd ACM Symp on Solid Modeling*, 1995.

[4] I. Fudos. Editable representations for 2d geometric design. Master's thesis, Purdue University, Dept. of Comp. Sci., 1993.

[5] I. Fudos and C. M. Hoffmann. Correctness proof of a geometric constraint solver. *Intl. J. of Computational Geometry and Applications*, page to appear, 1994.

[6] C. M. Hoffmann. Implicit curves and surfaces in CAGD. *IEEE Computer Graphics and Applications*, 13:79–88, 1993.

[7] Christoph M. Hoffmann and Pamela J. Vermeer. Geometric constraint solving in $R^2$ and $R^3$. In D. Z. Du and F. Hwang, editors, *Computing in Euclidean Geometry*. World Scientific Publishing, 1994. second edition.

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

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

[10] Wu Wen-Tsün. Basic principles of mechanical theorem proving in geometries. *J. of Systems Sciences and Mathematical Sciences*, 4:207–235, 1986.