An Approach to Geometric Constraint Solving for CAD Representations

Ioannis Fudos, Vicky Stamati and Antonis Protopsaltou
Department of Computer Science
University of Ioannina
GR45110 Ioannina, Greece
fudos@cs.uoi.gr

1. INTRODUCTION

A new generation of CAD systems has become available in which geometric constraints can be defined to determine properties of mechanical parts. The new design concept, often called *constraint-based design* or *design by features* [9, 17], offers users the capability of easily defining and modifying a design, but introduces the problem of solving complicated, not always well defined, constraint problems [3]. In this paper, we present the development of a user-friendly interactive system for editing and solving geometric configurations that arise in feature-based CAD/CAM systems. The system is built around a powerful graph-constructive constraint solving method presented in [14], capable of efficiently analyzing certain classes of *well-determined, over-determined* and *under-determined* configurations. Minimal systems of geometric constraints that are not solvable by the core constructive method are detected and may either be handled by a numerical method and treated afterwards as rigid bodies, or edited by the user. A main issue pertinent to geometric constraint solving is the solution selection problem. To this end, we have provided an interactive tool for navigating the constraint solver, to the intended solution. Consistent over-determined sub-configurations can be detected, interactively relaxed and solved appropriately. Under-determined subsystems are detected, isolated and subsequently presented to the user annotated with all possible constraint addition choices for interactive editing. To realize the constraint solver we have developed a prototype on a SUN workstation running Solaris 2.5.1. The graphical user interface was built in Java AWT, the core method was implemented in SETL (SET Language) as part of the work described in [5], the extensions for detecting underdetermined and over-determined configurations were also programmed in SETL, and for numerical solving MATLAB packages were invoked.

Section 2 provides an overview of methods for geometric constraint solving, and justifies the selection of the graph-constructive method of [14] for developing an interactive sketcher/solver appropriate for use in CAD / CAM systems. Section 3 briefly outlines the core graph-constructive method used. Section 4 presents the design and development of the graphical user interface, and the basic user interaction and system flow. Section 5 describes our experience with methods for treating over-determined and under-determined constraint configurations and their realization as part of our interactive software. Section 6 offers conclusions.

2. APPROACHES TO GEOMETRIC CONSTRAINT SOLVING

We present an overview of approaches to geometric constraint solving. We outline the most representative methods and evaluate their behavior in terms of the major concerns faced in CAD/CAM systems: solution selection, interactive speed, edit-ability, handling of over and underconstrained configurations and scope. A first version of this overview was presented in [16].

2.1 Numerical Constraint Solvers

In numerical constraint solvers, the constraints are translated into a system of algebraic equations and are solved using iterative methods. To handle the exponential number of solutions and the large number of parameters, iterative methods require sharp initial guesses. Also, most iterative methods have difficulties handling overconstrained or underconstrained instances. The advantage of these methods is that they have the potential to solve large nonlinear system that may not be solvable using any of the other methods. All existing solvers more or less switch to iterative methods when the given configuration is not solvable by the native method. This fact emphasizes the need for further research in the area of numerical constraint solving.

Sketchpad [31] was the first system to use the method of relaxation as an alternative to propagation. Relaxation is a slow but quite general method. The Newton-Raphson method has been used in various systems [24, 27], and it proved to be faster that relaxation but it has the problem that it may not converge or it may converge to an unwanted solution after a chaotic behavior. For that reason, Juno [24] uses as initial state the sketch interactively drafted by the user. However, Newton-Raphson is so sensitive to the initial guess [4], that the sketch drafted must almost satisfy all constraints prior to constraint solving. A sophisticated use of the Newton-Raphson method was developed in [22], where an improved way for finding the inverse Jacobian matrix is presented. Furthermore, the idea of dividing the matrix of constraints into submatrices as presented in the same work has the potential of providing the user with useful information regarding the constraint structure of the sketch. Though this information is usually quantitative and nonspecific, it may help the user in basic modifications. To check whether a constraint problem is well-constrained, Chyz [10] proposes a preprocessing phase where the graph of constraints is analyzed to check whether a necessary condition is satisfied. The method is however quite expensive in time and it cannot detect all the cases of singularity. An alternative method to Newton-Raphson for geometric constraint solving is homotopy or continuation [2], that is argued in [21] to be more satisfactory in typical situations where Newton-Raphson fails. Homotopy, is global, exhaustive and thus slow when compared to the local and fast Newton's method [23], however it may be more appropriate for CAD/CAM systems when constructive methods fail, since it may return all solutions if designed carefully.

2.2 Constructive Constraint Solvers

This class of constraint solvers is based on the fact that most configurations in an engineering drawing are solvable by ruler, compass and protractor or using other less classical repertoires of construction steps. In these methods the constraints are satisfied in a constructive fashion, which makes the constraint solving process natural for the user and suitable for interactive debugging. There are two main approaches in this direction.

Rule-constructive Solvers

Rule-constructive solvers use rewrite rules for the discovery and execution of the construction steps. In this approach, complex constraints can be easily handled, and extensions to the scope of the method are straightforward to incorporate [3]. Although it is a good approach for prototyping and experimentation, the extensive computations involved in the exhaustive searching and matching make it inappropriate for real world applications.

A method that guarantees termination, ruler and compass completeness and uniqueness using the Knuth-Bendix critical pair algorithm is presented in

[7, 28]. This method can be proved to confirm theorems that are provable under a given system of axioms [6]. A system based on this method was implemented in Prolog. Aldefeld in [1] uses a forward chaining inference mechanism, where the notion of direction of lines is imposed by introducing additional rules, and thus restricting the solution space. A similar method is presented in [32], where handling of overconstrained and underconstrained problems is given special consideration. Sunde in [30] uses a rule-constructive method but adopts different rules for representing directed and nondirected distances, giving flexibility for dealing with the solution selection problem. In [36], the problem of nonunique solutions is handled by imposing a topological order on three geometric objects. An elaborate description of a complete set of rules for 2D geometric constraint solving can be found in [34]. In their work, the scope of the particular set of rules is characterized. [20] presents an extension of the set of rules of [34], and provides a correctness proof based on the techniques of [13].

Graph-constructive Solvers

The *graph-constructive* approach has two phases. During the first phase the graph of constraints is analyzed and a sequence of construction steps is derived. During the second phase these construction steps are followed to place the geometric elements. These approaches are fast and more methodical. In addition, conclusions characterizing the scope of the method can be easily derived. A major drawback is that as the repertoire of constraints increases the graph-analysis algorithm needs to be modified.

Fitzgerald [12] follows the method of dimensioned trees introduced by Requicha [26]. This method allows only horizontal and vertical distances and it is useful for simple engineering drawings. .Todd in [33] first generalized the dimension trees of Requicha. Owen in [25] presents an extension of this principle that includes circularly'dimensioned sketches. -DCM [11] is a system that uses some extension of Owen's method. [14] presents an elaborative graph-constructive method, with fast analysis and construction algorithms, and extensions for handling classes of nonsolvable, underconstrained and consistently overconstrained configuration

2.3 Propagation Methods

Propagation methods follow the approach met in traditional constraint solving systems. In this approach, the constraints are first translated into a system of equations involving variables and constants. The equations are then represented by an undirected graph which has as nodes the equations, the variables and the constants, and whose edges represent whether a variable or a constant appears in an equation. Subsequently, we try to direct the graph so as to satisfy all the equations starting from the constants. To accomplish this, various propagation techniques have been used" but none of them guarantees to derive a solution and at the same time have a reasonable worst case running time. For a review of these methods see [28]. In a sense, the constructive constraint solvers can be thought of as a sub case of the propagation method (fixed geometric elements for constants and variable geometric elements for variables). However, constructive constraint solvers utilize domain specific information to derive more powerful and efficient algorithms.

2.4 Symbolic Constraint Solvers

In symbolic solvers, the constraints are transformed to a system of algebraic equations which is solved using methods from algebraic manipulation, such as Grabner basis calculation [8] or Wu's method [35]. Although, these methods are interesting from a theoretical viewpoint, their practical significance is limited, since their time and space complexity is typically exponential or even hyperexponential.

3. A GRAPH CONSTRUCTIVE METHOD

The method is presented in detail in [14]. We provide here a brief outline. A geometric constraint problem is given by a set of points, lines, rays, circles with prescribed radii, line segments and circular arcs, called the *geometric elements,* along with required relationships of incidence, distance, angle, parallelism, concentricity, tangency, and perpendicularity between any two geometric elements, called the *constraints.* The problem can be coded as a *constraint graph* G = *(V, E),* in which the graph nodes are the geometric elements and the constraints are the graph edges. The edges of the graph are labeled with the values of the distance and angle dimensions. Our constraint solving method first forms a number of rigid bodies[1] with three degrees of freedom, called *clusters.* For simplicity we will assume that a maximum number of clusters is formed, each cluster consisting of exactly two geometric elements between which there exists a constraint. Three clusters can be combined into a single cluster if they pairwise share a single geometric element. Geometrically, the combination corresponds to placing the associated geometric objects with respect to each other so that the given constraints can be satisfied. The constraint solving method works in two conceptual phases.

Phase 1 (analysis phase): The constraint graph is analyzed and a sequence of constructions is stipulated. Each step in this sequence corresponds to positioning three rigid geometric bodies (clusters) which pairwise share a geometric element (point or line).

Phase 2 (construction phase): The actual construction of the geometric elements is carried out, in the order determined by Phase 1, by solving certain standard sets of algebraic equations.

[1] A rigid body is a set of geometric elements whose position and orientation relative to each other is known.

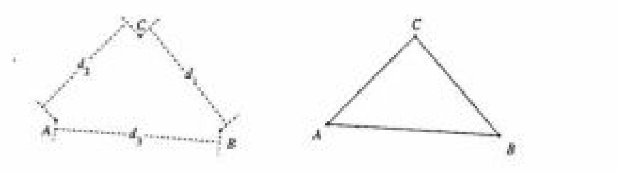


Figure 1. Constraint problem (left), and associated constraint graph (right).

To illustrate the process, consider three points *A, B,* and *C* between which distances have been prescribed, as shown in Figure 1 left. The associated constraint graph is shown on the right. In Phase 1 of the constraint solving, we determine first that every pair of points can be constructed separately, resulting in three clusters. Moreover, the three clusters can be combined into a single cluster since they share pairwise a geometric element. The combination merges the three clusters into one. As soon as a single cluster is obtained, Phase 1 considers the constraint problem *solvable,* Phase 1, the analysis phase, consists of two parts:

- the *reduction analysis* that produces a sequence of local cluster merges and handles well-constrained and overconstrained problems, and

- the *decomposition analysis* that produces a sequence of decompositions (that correspond to a reverse sequence of duster merges) and handles under constrained cases, The outcome of the reduction analysis is fed as input to the decomposition analysis.

4. USER INTERACTION AND SYSTEM FLOW

We have developed a graphical user interface that integrates a geometric constraint solver with an editor for constructing and modifying dimensioned sketches representing CAD cross-sections [9]. For the development of the graphical user interface we used Java's AWT (Advanced Window Toolkit), for three main reasons:

- We were interested in building a rapid prototype to test and tune the effectiveness of our interactive method, Java's AWT provides a relatively easy way for producing graphical user interfaces. Moreover, the new version of AWT has additional flexibility and an improved method for relating user triggered events with actions. These features made AWT a good candidate for developing the GUI of our interactive solver.
- Although executing Java code induces a significant overhead in terms of CPU and memory consumption, this overhead is not noticeable in workstations with sufficient memory.
- Producing Java code makes our graphical user interface portable to the vast majority of platforms.

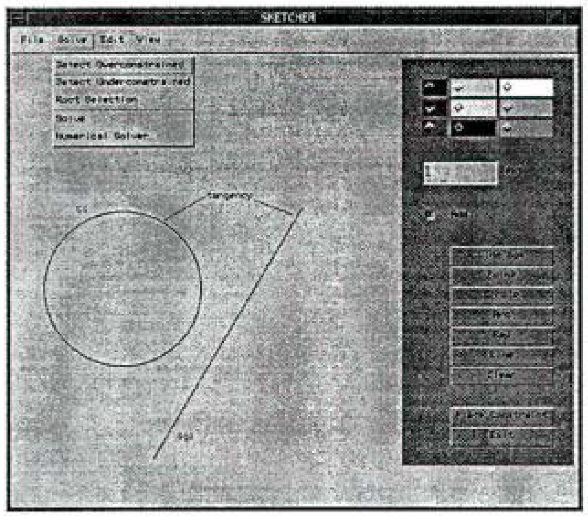
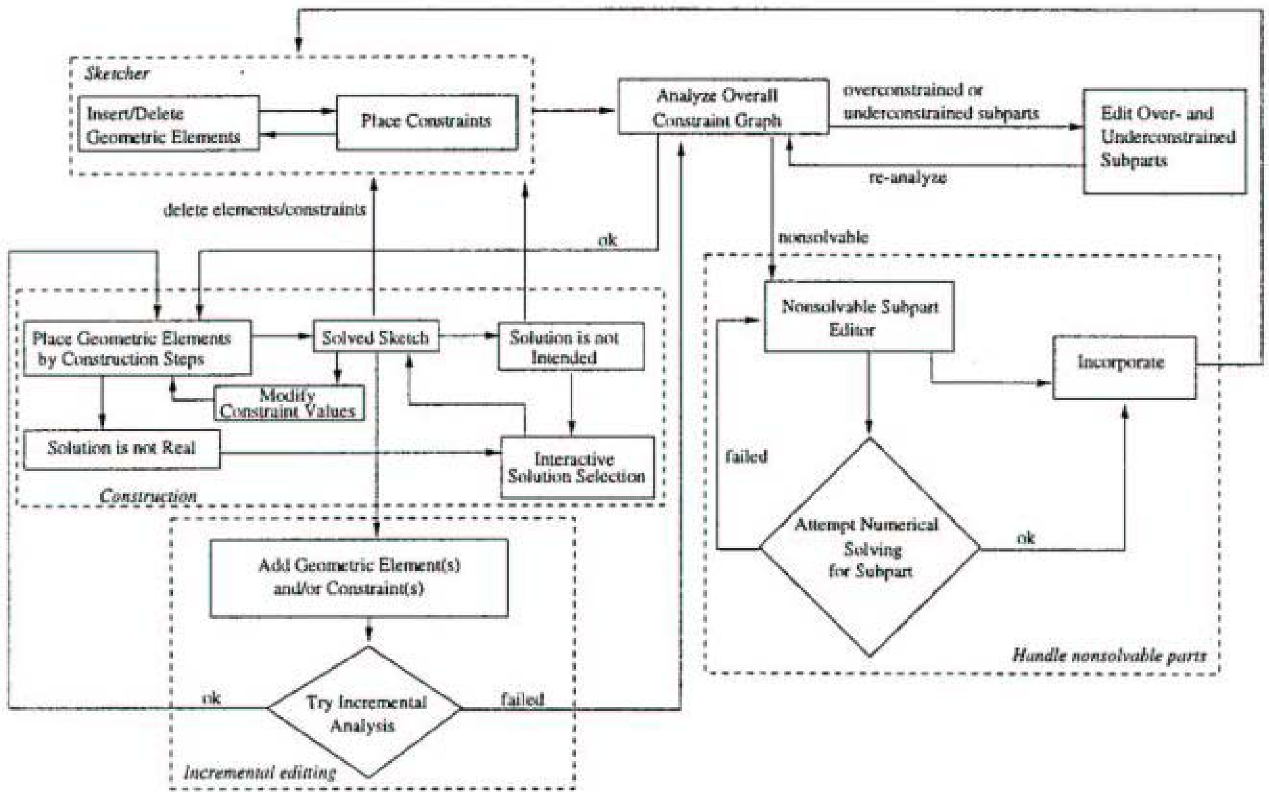


Figure 2. The Graphical User Interface.

The overall user-system interaction is as follows. The user draws a sketch and then imposes constraints on the sketched elements. The user can then invoke the solver which will try to solve the system of geometric constraints and return a new sketch with the geometric elements placed appropriately. If the solver is incapable of solving completely the sketch the user is notified and certain actions may be taken to correct the source of the problem. After successfully solving a dimensioned sketch the user may add/delete geometric objects and constraints, and re-invoke the solver. We have implemented points, circles, rays, lines, line segments and arcs. The available constraints are distance, tangency, angle, coincidence, concentricity and constraints that can be expressed as a combination of the above. We have also included text, for user annotation, in the form of constrained rectangles.

In Figure 2, a snapshot of the graphical user interface is depicted. The menu for inserting geometric objects and text lays on the right hand side of the main window. At the bottom of this menu above the Exit button there is the the Place Constraint button that transforms the functionality of the right-hand side menu for adding geometric constraints. At the top side there are four pull-down menus. The `File` menu has the `Save/Load` functions, and some auxiliary ones such as `Merge` and `Quit`. The files contain description of geometric objects, constraints, display information and annotation expressed in a language similar to the one presented in [15]. The communication with the main solver program and the numerical solver is realized through files with the same content augmented with solution specific information from the solver to the user and vice versa. The processing of the files is made possible by translating the high level language to an easy to read list of values, by using a compiler developed in C, using the flex and bison tools.

The `View` menu hides or makes visible certain classes of objects such as constraints, geometric elements, and annotation. The Edit menu contains the basic `Delete`, `Move` and `Copy` operations which are applicable to both geometric objects and constraints.



Sketcher
Insert/Delete Geometric Elements
Place Constraints
Analyze Overall Constraint Graph
overconstrained or underconstrained subparts
Edit Over- and Underconstrained Subparts
re-analyze
delete elements/constraints
ok
nonsolvable
Place Geometric Elements by Construction Steps
Solved Sketch
Solution is not Intended
Modify Constraint Values
Solution is not Real
Interactive Solution Selection
Construction
Nonsolvable Subpart Editor
Incorporate
failed
Attempt Numerical Solving for Subpart
ok
Handle nonsolvable parts
Add Geometric Element(s) and/or Constraint(s)
ok
Try Incremental Analysis
failed
Incremental editing

Figure 3. Interaction Flow for Solving a Sketch.

The `Solve` menu is illustrated in Figure 2. It contains the basic functions for performing geometric constraint solving of the given dimensioned sketch. Figure 3 shows the overall interaction flow for solving a sketch. The user first attempts to solve the sketch by the extended graph-constructive method outlined in Section 3. This is performed by choosing `Solve` from the `Solve` menu. If the analysis finds a valid sequence of construction steps for placing all geometric elements, this sequence is followed for producing the final Solved Sketch. However, if the solution does not consist of real numbers, or is not the one intended by the user, an interactive tool may be used for navigating the solver to a meaningful solution. The user is presented with the construction sequence and is given the capability of modifying the relative positioning of the objects involved, thus affecting stepwise the overall solution selection. Changing the values of some constraints (e.g distances, angles), of a solved sketch will result in re evaluating the placement steps (not the graph analysis). When the method cannot solve the configuration, an error message is returned and the user is provided with an enhanced right-hand side menu for interactive intervention. Using this menu, the user may browse the part(s) of the design that correspond(s) to the subgraph(s) that is(are) not solvable. Finding minimal well-constrained subparts is a difficult process that has a worst case time complexity *O(n4)* in our implementation. A more sophisticated method *O(n2)* has been proposed in [18], but it may need some care to to be applied to rigid bodies without spoiling the quadratic complexity. The nonsolvable parts can be edited by deleting and adding constraints and geometric objects. These modifications are incorporated to the overall design by pressing the `Incorporate` button at the enhanced right-hand side menu. This drives the design process back to the first step. Currently, the `Text, Copy` and `Move` buttons are disabled during the process of editing the nonsolvable parts of the design. Another option for handling nonsolvable parts is to invoke the Numerical Solver for one or more of the nonsolvable parts. If this succeeds, the parts are subsequently treated as rigid bodies. To support this feature we have defined a rigid body structure in the representation language. If the basic solver detects overconstrained or underconstrained configurations it notifies the user. The process for treating such configurations is described in more detail in the next section.

Finally, the user may choose to incrementally edit the solved sketch by adding constraints and geometric elements. Invoking `solve` in this case tries to incrementally solve the new design. If incremental solving fails, analyzing the overall system of constraints is attempted. Deleting a geometric element or constraint is not considered an incremental change and analysis of the overall graph is attempted.

5. HANDLING OVER AND UNDER-DETERMINED CONFIGURATIONS

Each line or point on the Euclidean plane has two degrees of freedom. Each distance or angle corresponds to one equation. If there are no fixed geometric elements (i.e., geometric elements whose absolute coordinates have been specified explicitly by the user), then we expect that, $|E| = 2|V| - 3$, where $|V|$ is the number of geometric elements and $|E|$ is the number of constraints. This holds for lines and points with distances and angles constraints only. When other objects or constraints exist, we have to substitute with sums for the degrees of freedom and for the constraints. Note that the solution will be a rigid body with three remaining degrees of freedom, because the constraints determine only the relative position of the geometric elements. An example is shown in Figure 4. In the figure, the vertex P of the quadrilateral has a well-defined position when $\alpha+\beta=90^0$. But for $\alpha+\beta\neq90^0$the position of P is not determined. This "semantic" notion of well-constrained problems can be made specific for the constraint graph analysis, because there the generic problem of constructing a solution is considered independently of dimension values. Intuitively, a dimensioned sketch is considered to be well-constrained, if it has a finite number of solutions for nondegenerate configurations. Similarly, a dimensioned sketch is considered to be underconstrained, if it has an infinite number of solutions for nondegenerate configurations. Finally, a dimensioned sketch is considered to be overconstrained, if it has no solutions for nondegenerate configurations. The intuitive notions above can be made technically precise for the euclidean plane (see e.g., [13]). For the 3D case, however, the necessary and sufficient conditions for specifying whether a configuration is well-constrained are not known.

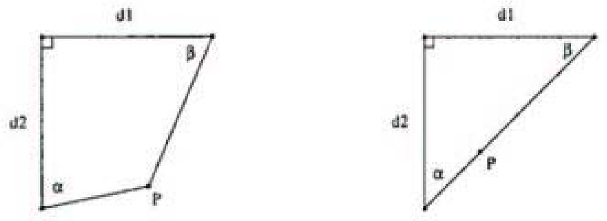


Figure 4. Degenerate Configuration (right) for $\alpha+\beta=90^0$ **.**

For an algorithm that tests whether a graph is structurally well-constrained in 2D see e.g. [19, 29]. Note that a structurally well-constrained graph can be overconstrained in a geometric sense, for example if there are three lines with pairwise angle constraints. The core reduction analysis handles structurally well-constrained and overconstrained problems. The decomposition analysis handles structurally underconstrained problems. Thus, the graph analysis may succeed to produce a sequence for placement, even for overconstrained or underconstrained configurations. In any case, the user is notified about the existence of ill-determined configurations. The user may then use the `Detect Underconstrained` and `Detect Overconstrained` buttons to analyze the graph. To detect the overconstrained subgraph, the system executes a reduction analysis and keeps track of cluster merging that states the existence of overconstrained configurations. Then, the user is presented with the corresponding part that is overconstrained, after running a simple algorithm that marks constraints that can be candidates for elimination. To detect the underconstrained, the result of the decomposition analysis is run, and the plausible constraint additions are highlighted.

6. CONCLUSIONS

We have presented an overview of approaches to geometric constraint solving and selected a powerful graph-constructive method for building around it an interactive system for designing and solving dimensioned sketches. We have presented an innovative paradigm of user interaction for using the core method, treating ill-determined configurations, handling nonsolvable cases and performing solution selection. We have developed a prototype to evaluate this paradigm and to fine tune the system-user interaction.

7. ACKNOWLEDGMENTS

This work is supported by a Research Grant (PENED – 01ED329) by the General Secretariat of Research and Technology, Greece.

8. REFERENCES

[1] B. Aldefeld. Variation of geometries based on a geometric-reasoning method. Computer Aided Design, 20(3):117-126, April 1988.

[2] E. L. Allgower and K. Georg. Continuation and path following. Acta Numerica, pages 1-64, 1993.

[3] Bruderlin and D. Roller (eds). Geometric Constraint Solving and Applications.Springer Verlag, 1998.

[4] Paula L. Beaty, Patrick A. Fitzhorn, and Gary J. Herron. Extensions in variational geometry that generate and modify object edges composed of rational Bezier curves. Computer Aided Design, 26(2):98-107, 1994.

[5] W. Bouma, 1. Fudos, C. M. Hoffmann, J. Cai, and R. Paige. A Geometric Constraint Solver. Computer Aided Design, 27(6):487-501, June 1995.

[6] B. Bruderlin. Using geometric rewrite rules for solving geometric problems symbolically. Theoretical Computer Science, 116:291-303, 1993.

[7] Beat Bruderlin. Constructing Three-Dimensional Geometric Objects Defined by Constraints. In Workshop on Interactive 3D Graphics, pages 111-129. ACM, October 23-24 1986.

[8] B. Buchberger. Grobner Bases: An Algorithmic Method in Polynomial Ideal Theory. In N. K. Bose, editor, Multidimensional Systems Theory, pages 184-232. D. Reidel Publishing Company, 1985.

[9] X. Chen and C. M. Hoffmann. On Editability of Feature Based Design. Computer Aided Design, 27:905-914, 1995.

[10] W. Chyz. Constraint management for CSG. Master's thesis, MIT, June 1985.

[11] D-Cubed Ltd, 68 Castle Street, Cambridge, CB3 OAJ, England. The Dimensional Constraint Manager, June 1994. Version 2.7.

[12] W. Fitzerland. Using Axial Dimensions to Determine the Proportions of Line Drawings in Computer Graphics. Computer Aided Design, 13(6), November 1981.

[13] Fudos and C. M. Hoffmann. Correctness Proof of a Geometric Constraint solver. International Journal of Computational Geometry 8 Applications, 1995.

[14] Fudos and C. M. Hoffmann. A Graph-constructive Method to Solving systems of Geometric Constraints. ACM Transactions on Graphics, 16(2):179-216, 1997.

[15] Ioannis Fudos. Editable Representations for 2D Geometric Design. Master's thesis, Dept of Computer Sciences, Purdue University, December 1993.

[16] Ioannis Fudos. Constraint Solving for Computer Aided Design. PhD thesis, Department of Computer Sciences, Purdue University, August 1995.

[17] C. M. Hoffmann and R. Joan. On User-Defined Features. Computer Aided Design, 30:321-332, 1998.

[18] C. M. Hoffmann, A. Lomonosov, and M. Sitharam. Finding Solvable Subsets of Constraint Graphs. In G. Smolka, editor, LNCS 1330, pages 463-477. Springer Verlag, 1997.

[19] H. Imai. On combinatorial structures of line drawings of polyhedra. Discrete and applied Mathematics, 10:79, 1985.

[20] R. Juan-Arinyo and Antoni Soto. A rule-constructive geometric constraint solver. Technical Report LSI-95-25-R, Universitat Politecnica de Catalunya, 1995.

[21] H. Lamure and D. Michelucci. Solving geometric constraints by homotopy. In Proc. Third Symposium on solid Modeling and Applications, pages 263-269, Salt Lake City, 1995. ACM.

[22] Robert Light and David Gossard. Modification of geometric models through variational geometry. Computer Aided Design, 14(4):209-214, July 1982.

[23] A. Morgan. Solving polynomial systems using continuation for engineering and scientific problems. Prentice-Hall, Inc., 1987.

[24] G. Nelson. Juno, a costraint-based graphics system. In SIGGRAPH, pages 235-243, San Francisco, July 22-26 1985. ACM.

[25] J. C. Owen. Algebraic Solution for Geometry from Dimensional Constraints. In ACM Symp. Found. of Solid Modeling, Austin, TX, pages 397-407. ACM, 1991.

[26] A. Requicha. Dimensionining and tolerancing. Technical report, Production Automation Project, University of Rochester, May 1977. PADL TM-19.

[27] D. Serrano and D. Gossard. Combining mathematical models and geometric models in CAE systems. In Proc. ASME Computers in Eng. Conf., pages 277-284, Chicago, July 1986. AS ME.

[28] Wolfang Sohrt. Interaction with Constraints in three-dimensional Modeling. Master's thesis, Dept of Computer Science, The University of Utah, March 1991.

[29] K. Sugihara. Detection of Structural Inconsistencies in Systems of Equations with Degrees of Freedom and its Applications. Discrete Applied Mathematics, 10:297-312, 1985.

[30] Geir Sunde. Specification of shape by dimensions' and other geometric constraints. In M. J. 'Wozny, H. W. McLaughlin, and J. L. Encarnacao, editors, Geometric modeling for CAD applications, pages 199-213. North Holland, IFIP, 1988.

[31] I. Sutherland. Sketchpad, a man-machine graphical communication system. In Proc. of the spring Joint Compo Conference, pages 329-345. IFIPS, 1963.

[32] Hirimasa Suzuki, Hidetoshi Ando, and Fumihiko Kimura. Variation of geometries based on a geometric-reasoning method. Comput. & Graphics, 14(2):211-224, 1990.

[33] Philip Todd. A k-tree generalization that characterizes consistency of dimensioned engineering drawings. SIAM J. DISC. MATH., 2(2):255-261, 1989.

[34] A. Verroust, F. Schonek, and D. Roller. Rule-oriented method for parameterized computer-aided design. Computer Aided Design, 24(3):531-540, October 1992.

[35] Wu Wen- Tsun. Basic Principles of Mechanical Theorem Proving in Elementary Geometries. Journal of Automated Reasoning, 2:221-252, 1986.

[36] Yasushi Yamaguchi and Fumihiko Kimura. A constraint modeling system for variational geometry. In M. J. Wozny, J. U. Turner, and K. Preiss, editors, Geometric Modeling for Product Engineering, pages 221-233. Elsevier Science Publishers B.V. (North Holland), 1990.