A brief on constraint solving
Hoffmann, C.M. and Joan-Arinyo, R.
Research Report LSI-04-49-R
A Brief on Constraint Solving

Christoph M. Hoffmann
Department of Computer Science
1398 Computer Science Building
Purdue University
West Lafayette, IN 47907-1398, USA

Robert Joan-Arinyo
Escola Tècnica Superior d’Enginyeria Industrial
Universitat Politècnica de Catalunya
Av. Diagonal 647, 8a, E-08028 Barcelona

October 8, 2004

Abstract

We survey the current state of the art in geometric constraint solving. Both 2D and 3D constraint solving is considered, and different approaches are characterized.

Keywords: Geometric constraints, constraint solving, parametric design.

1 Introduction and Scope

2D geometric constraint solving is arguably a core technology of computer-aided design (CAD) and, by extension, of managing product design data. Since the introduction of parametric design by Pro/Engineer in the 1980s, every major CAD system has adopted geometric constraint solving into its design interface. Most prominently, 2D constraint solving has become an integral component of sketchers on which most systems base feature design.

Beyond applications in CAD and, by extension, in manufacturing, geometric constraint solving is also applicable in virtual reality and is closely...
related in a technical sense to geometric theorem proving. For solution techniques, geometric constraint solving also borrows heavily from symbolic algebraic computation and matroid theory.

In this paper, we review basic techniques that are widely available for solving 2D and 3D geometric constraint problems. We focus primarily on the basics of 2D solving and touch lightly on spatial constraint solving and the various ways in which geometric constraint solvers can be extended with relations, external variables, and parameter value enclosures. These and other extensions and problem variants have been published in the literature. They are recommended to the interested reader as follow-on material for study.

2 Preliminaries

A geometric constraint problem can be characterized by means of a tuple \((E, O, X, C)\) where

- \(E\) is the geometric space constituting a reference framework into which the problem is embedded. \(E\) is usually Euclidean.
- \(O\) is the set of specific geometric objects which define the problem. They are chosen from a fixed repertoire including points, lines, circles and the like.
- \(X\) is a, possibly empty, set of variables whose values must be determined. In general, variables represent quantities with geometric meaning: distances, angles and so on. When the quantities are without a geometric meaning, for example, when they quantify technological aspects and functional capabilities, those variables are called external.
- \(C\) is the set of constraints. Constraints can be geometric or equational. Geometric constraints are relationships between geometric elements chosen from a predefined set, e.g., distance, angle, tangency, etc. The relationship (the distance, the angle, ...) is represented by a tag. If the tag represents a fixed value, the constraint is called valued. If the tag represents a value to be computed, the constraint is called symbolic, [32].

Equational constraints are equations some of whose variables are tags of symbolic constraints. The set of equational constraints can be
empty.

The geometric constraint solving problem can now be stated as follows:

Given a geometric constraint problem \((E, O, X, C)\),

1. Are the geometric elements in \(O\) placed with respect to each other in such a way that the constraints in \(C\) are fulfilled? If the answer is positive, then

2. given an assignment of values to the valuated constraints and external variables, is there an actual construction that fulfills the constraints and equations?

When dealing with geometric constraint solving, the first issue that needs to be settled is the dimension of the embedding space \(E\). In 2D Euclidean space, \(E = \mathbb{R}^2\), a number of techniques have been developed that successfully solve the geometric constraint solving problem. For an in-depth review see Jermann, [41]. However, there remain open questions such as characterizing the competence (also called domain) of the known techniques.

Spatial constraint solving, where \(E = \mathbb{R}^3\), include problems in fields like molecular modeling, robotics, and terrain modeling. Here, both a good conceptualization and an effective solving methodology for the geometric constraint problem has proved to be difficult. Pioneering work has been reported by Hoffmann and Vermeer, [37, 38] and by Durand, [18].

2.1 The General Problem

Figure 1 depicts a general geometric constraint solving problem in \(\mathbb{R}^2\). Problem components are

- The set of geometric elements, \(O = \{A, B, C, D, L_{AB}, L_{AC}, L_{BC}\}\).
- The set of tags in the constraints, \(P = \{d, h, \alpha\}\).
- The set of geometric variables, \(V_1 = \{x, y\}\).
- The set of external variables, \(V_2 = \{v\}\).
- The set of geometric constraints with fixed tags, \(C_1 = \{dpp(A, B) = d, \ dpl(C, L_{AB}) = h, \ ang(L_{AB}, L_{BC}) = \alpha\}\).
Figure 1: The general geometric constraint solving problem. Example in $\mathcal{R}^2$.

- The set of geometric constraints with variable tags, $C_2 = \{dpp(A, C) = x, \ dpp(C, D) = y\}$.
- The set of equational constraints, $C_3 = \{y = x \cdot v, \ v = 0.5 \cos(\alpha)\}$.

with $X = V_1 \cup V_2$ and $C = C_1 \cup C_2 \cup C_3$.

Presented in this way, the geometric constraint solving problem involves in general issues concerning how to deal with external variables. Here we refer the interested reader to the work by Hoffmann and Joan-Arinyo, [32], and Joan-Arinyo and Soto, [46].

2.2 The Basic Problem

The basic constraint problem only considers geometric elements and constraints whose tags are assigned a value. It excludes external variables, constraints whose tags must be computed, and equational constraints. So the basic problem is stated in the following way.

Given a set $O$ with $n$ geometric elements and a set $C$ with $m$ geometric constraints defined on them

1. Is there a placement of the $n$ geometric elements such that the $m$ constraints are fulfilled? If the answer is positive,

2. given an assignment of values to the $m$ constraints tags, is there an actual construction of the $n$ geometric elements fulfilling the constraints?
Figure 2: Piston, crankshaft and connecting rod mechanism.

Figure 2 depicts a piston-crankshaft mechanism, [17], a basic geometric constraint solving problem. The left side shows the problem, the right side shows the actual mechanism. The mechanism transforms the translational motion of point \( p_5 \) along the straight line \( l_1 \) into a rotational motion of point \( p_4 \), on a circular path with center \( p_3 \) and radius \( d_3 \).

The piston-crankshaft mechanism can be abstracted as a geometric constraint solving problem comprising five points \( p_i \), \( 1 \leq i \leq 5 \), and a straight line \( l_1 \). The set of constraints is given in Figure 3 and includes point-point distances, \( dpp(\cdot) \), and coincidences, \( on(\cdot) \).

In what follows we will focus on the basic geometric constraint solving problem.

2.3 Problem Categorization

The CAD/CAM community focuses on the design and manufacture of rigid objects, that is, objects that are fully determined up to a global coordinate system. Similarly, we seek solutions to a constraint problem that are determined up to a global coordinate system, that is, where solutions are congruent under the rigid-body transformations of translation and rotation. We call a configuration of geometric objects rigid when all objects are fixed with respect to each other up to translation and rotation. For simplicity, we call such a configuration \textit{rigid}.

An intuitive way to introduce rigidity comes from considering the number
1. \( \text{dpp}(p_1, p_2) = d_1 \)
2. \( \text{dpp}(p_2, p_3) = d_2 \)
3. \( \text{dpp}(p_3, p_4) = d_3 \)
4. \( \text{dpp}(p_4, p_5) = d_4 \)
5. \( \text{dpp}(p_1, p_5) = d_5 \)
6. \( \text{on}(p_1, l_1) \)
7. \( \text{on}(p_2, l_1) \)
8. \( \text{on}(p_3, l_1) \)
9. \( \text{on}(p_5, l_1) \)

Figure 3: A set of geometric constraints for the piston-crankshaft mechanism.

of solutions that a geometric constraint problem has. There are three categories: A problem is \textit{structurally under constrained} if there are infinitely many solutions that are not congruent under rigid transformation, \textit{structurally well-constrained}, if there are finitely many solutions modulo rigid transformation, and \textit{structurally over constrained} if the deletion of one or more constraints results in a well-constrained problem. A constraint problem naturally corresponds to a set of (usually nonlinear) algebraic equations.

Defined in this way, the concept of rigidity appears to be simple but it is not quite in accord with the intuition about rigidity. The categories so defined only refer to the problem’s structure and do not account for other issues such as, for example, inconsistencies that could originate from specific values assigned to the constraints. Clearly a problem that is structurally well-constrained could actually be underconstrained for specific values of the constraints.

For example, consider the structurally well constrained problem given in Figure 4, see Fudos and Hoffmann, [22]. Point \( P \) is properly placed whenever \( \alpha + \beta \neq 90^\circ \) and the problem is well-constrained. But if \( \alpha + \beta = 90^\circ \), then the placement for point \( P \) is undetermined and, therefore, the problem is no longer well constrained.

Different formal definitions of rigidity have been explored in the literature. See, for example, the work by Henneberg, [26], and Laman, [59], or the more recent works by Graver et al. [24], Fudos and Hoffmann, [22], Hoffmann et al., [33], and Whitley, [87, 88].

3 General Properties of Solving Techniques

The properties a geometric constraint solver should have include
Figure 4: Left: General configuration. Right: Degenerate configuration for $\alpha + \beta = 90^\circ$.

- **Soundness**: The solver always halts and, if a solution is found, it satisfies all stipulated constraints.

- **Completeness**: The solver solves all solvable problems and announces the unsolvability of all unsolvable problems.

- **Competence**: Since the known, efficient methods are not complete, a related property that is more reasonable in practice is that the solver solves all solvable problems in a subdomain of interest and, for unsolvable problems in the subdomain, announces unsolvability.

- **Persistence**: With the same set of (valuated) constraints the solver always finds the same solution.

- **Stability**: Under small changes of assignments to the geometric constraint tags, the solution found is nearby.

- **Efficiency**: Efficiency measures the computational cost, that is, whether solutions are found rapidly.

- **Robustness**: A solver is robust if the solutions are not adversely affected by the finite precision of floating-point computations.

Additional desirable properties that enhance a solver’s performance are

- **Intensionality**: The solver finds solutions that the user is interested in.
• **Geometric sense:** A solver has geometric sense whenever the solutions it finds can be expressed as a sequence of geometric construction steps.

• **Dimension independence:** The solver can be used for problems embedded in the space $E = \mathbb{R}^n$, independent of $n$.

• **Generality:** The solver can deal with the general geometric constraint solving problem including symbolic constraints and external variables.

For 2D solvers, there are good compromises that achieve a reasonable degree of these solver characteristics. However, it is difficult to obtain solvers that exhibit persistence and stability fully, [54].

In general, solver competence is the antithesis of efficiency since constraint solving is of doubly exponential complexity. Intensional problems include root selection, orientation, and topological degeneracy for specific dimensional values. They may look simple but generally are associated with complex mathematical problems.

4 Major Approaches

Geometric constraint solving methods can be roughly classified as graph-based, logic-based, or algebraic. For 2D solvers, the graph-based approach has become dominant in CAD. A problem closely related to geometric constraint solving is Automated Theorem Proving. We will make a few remarks about it.

4.1 Graph-Based Approach

In the graph-based approach, the constraint problem is translated into a graph (or hyper graph) whose vertices represent the geometric elements and whose edges the constraints upon them. See Section 9. The solver analyzes the graph and formulates a solution strategy whereby subproblems are isolated and their solutions suitably combined. A subsequent phase then solves all subproblems and combines them. The advantage of this type of solver is that the subproblems often are very small and fall into a few simple categories. The disadvantage is that the graph analysis of a fully competent solver is rather complicated.
The graph-based approach can be further subdivided as constructive, degree of freedom analysis, and propagation.

4.1.1 Constructive Approach

The constructive approach generates the solution to a geometric constraint problem as a symbolic sequence of basic construction steps. Each step is a rule taken from a predefined set of operations that position a subset of the geometric elements. For example, the operations may restrict to ruler-and-compas constructions. Clearly, this approach preserves the geometric sense of each operation involved in the solution. Note that the sequence of construction steps allows to compactly represent a possibly doubly exponential number of solution instances. However, the constructive approach cannot solve problems with symbolic constraints or external variables.

Depending on the technique used to analyze the problem, two different categories of constructive approaches can be distinguished: top-to-bottom and bottom-up.

The top-to-bottom technique recursively splits the problem until it has isolated simpler, basic problems whose solutions are known. In this category, Todd, [83], defines the r-tree concept and derives a geometric constraint solving algorithm. Owen, [71], describes a more general method based on the recursive decomposition of the constraints graph into triconnected components. Inspired by Owen's work, Fudos reported a new decomposition method in [20, 22]. Efficient algorithms with a running time $O(n^2)$, where $n$ is the number of geometric elements, are known for the methods of Owen, [71], and Fudos, [20, 22].

In the bottom-up approach, the solution is built by suitably combining recursively solutions to subproblems already computed, starting from the constraints in the given set, considering each constraint as a single element.

Usually, constraints are represented implicitly as a collection of sets of geometric elements where the elements of each set are placed with respect to a local framework. Sets are merged by application of rewriting rules until all the geometric elements are included in just one set. The advantage of this representation is that the sets of constraints capture the relationships between geometric elements compactly. Fudos et al., in the method described in [6, 20, 21, 22], use one type of sets of constraints, called cluster, and one generic rule that merges three clusters which pairwise share two elements.
Lee et al., [62], describe a constructive method that associates with each vertex in the graph a status which can be defined, half defined or not defined. Inference rules are used to modify the status of the vertices.

Efficient algorithms with a running time $O(n^2)$, where $n$ is the number of geometric elements, are known for the methods listed above. However, the constructive approach is not complete, therefore assessing the competence of solvers in this category is an important issue. Verrout, [85, 86], partially characterizes the set of relevant problems solved by the solver described. Joan-Arinyo et al., [47, 48], describe a formalization that unifies the methods reported by Fudos, [21, 22], and Owen, [71]. In [47, 49], Joan-Arinyo et al. show that the sets of problems solved by Fudos’ and Owen’s approaches are the same.

In [32] Hoffmann and Joan-Arinyo describe a technique that extends constructive methods with the capability of managing functional relationships between geometric and externals variables. Essentially, the technique combines two methods: one is a constructive constraint solving method, and the other is a systems of equations analysis method. Joan-Arinyo and Soto-Riera, [46], further improved the technique and formalized it as a rewriting system.

Constructive methods essentially work in 2-space. However, several attempts to extend them to 3-space have been reported. See, for example, Brüderlin [7], Verrout [85], and Hoffmann and Vermeer [37, 38].

4.1.2 Degrees of Freedom Analysis

Degrees of Freedom Analysis assigns degrees of freedom to the geometric elements by labeling the vertices of the graph of the problem. Each edge of the graph is labeled with the number of degrees of freedom canceled by the associated constraint. Then the method solves the problem by analyzing the resulting labeled graph.

Kramer, [55, 56, 57], developed a method to solve specific problems from the kinematics of mechanisms. The method applies techniques borrowed from the process planning field to yield a symbolic solution. Since the set of rules used to generate the plan preserves geometric sense, the entire method also preserves it. Kramer, [55], proves that his method is correct by showing that the set of rules together with the labeled graph is a canonical rewriting system. The method runs in time $O(nm)$, where $n$ is the number of geomet-
ric elements and \( m \) the number of constraints in the problem. Since \( m \) is typically \( O(n) \), the method has the same complexity as the constructive approach. Bhansali et al., [3], describe a method that generates automatically segments of the symbolic solution in Kramer’s approach.

Salomons et al., [73], represent objects and constraints as a graph and apply geometric and equational reasoning following the lines given by Kramer’s method.

In [39, 40], Hsu reports a method with two phases. First, a symbolic solution is generated. Then, the actual construction is carried out. The method applies geometric reasoning and, if this fails, numerical computation.

Latham et al., [61], decompose the labeled graph in minimal connected components called balanced sets. If a balanced set corresponds to one of the predefined specific geometric constructions, then it can be solved. Otherwise the underlying equations are solved numerically. The method also deals with symbolic constraints and identifies over- and under constrained problems. Assigning priorities to the constraints allows them to solve over constrained problems. A proof of correctness is given in [61].

Recently, Hoffmann et al., [33, 34, 35, 36], have developed a flow-based method for decomposing graphs of geometric constraint problems. The method generically iterates to obtain a decomposition of the underlying algebraic system into small subsystems called minimal dense subgraphs. The method fully generalizes degree-of-freedom calculations, other approaches based on matching specific subgraphs patterns, as well as prior flow-based approaches. However, the decomposition rendered does not necessarily have geometric sense since minimal dense subgraphs can be of arbitrary complexity far exceeding problems considered by geometric construction.

4.1.3 Propagation Approach

Propagation methods represent the set of algebraic equations with a symmetric graph whose vertices are variables and equations and whose edges are labeled with the occurrences of the variables in the equations.

Propagation methods try to orient the edges in the graph in such a way that each equation vertex is a sink for all the edges incident on it except one. If the process succeeds, then there is a general incremental solution. That is, the system of equation can be transformed into a triangular system and solved using back substitution.
Among the techniques to orient a graph we find in the literature degrees of freedom propagation and propagation of known values, [19, 74, 84]. Propagation methods do not guarantee finding a solution whenever one exists. When they fail, for example, when the orientation algorithm creates a loop, propagation methods are combined with numerical methods, [5, 58, 78, 82]. Veltkamp and Arbab, [84], apply other techniques to break loops created while orienting the graph.

Leler, [63], describes propagation methods in depth and reports on a new tool named augmented rewriting terms which consists of a classical rewriting system along with an association of atom-value and object-type. This tool has had success in resolving certain systems of nonlinear equations.

In [4], Borning et al., describe an local propagation algorithm that can deal with inequalities.

4.2 Logic-Based Approach

In the logic-based approach, the problem is translated into a set of assertions and axioms characterizing the constraints and the geometric objects. By employing reasoning steps, the assertions are transformed in ways that expose solution steps in a stereotypical way and special solvers then compute coordinate assignments.

Akelefeld, [1], Brüderlin, [8, 9, 10], Sohrt, [76, 77], Sohrt, et al., [76, 77], and Yamaguchi et al., [91], use first order logic to derive geometric information applying a set of axioms from Hilbert's geometry. Essentially these methods yield geometric loci at which the elements must be.

Sunde, [81], and Verroust, [85, 86], consider two different types of sets of constraints: sets of points placed with respect to a local framework, and sets of straight line segments whose directions are fixed with respect to a local framework. The reasoning is basically performed by means of a rewriting system on the sets of constraints. The problem is solved when all the geometric elements belong to a unique set. Joan-Arinyo and Soto-Riera, [44, 45], extended these sets of constraints with a third type consisting of sets containing one point and one straight line such that the perpendicular point-line distance is fixed.
4.3 Algebraic Methods

In the algebraic approach, the constraint problem is translated into a set of nonlinear equations and is then solved using any of the available methods for solving nonlinear equations. The main advantages of algebraic solvers are their generality, dimension independence and the ability to deal with symbolic constraints naturally.

In principle, an algebraic solver can be fully competent. However, algebraic solvers may have low efficiency or may have difficulty constructing solutions reliably. When used to pre-process and study specific constraint systems, algebraic techniques can be extremely useful and very practical.

As a result of mapping the geometric domain problem into an equational one, the geometric sense of the solutions rendered is lost. Moreover, well constrained problems are mapped to under constrained systems of equations because constraints fix the placement for each geometric element with respect each other modulo translation and rotation. Therefore a set of additional equations to cancel these degrees of freedom must be added.

Algebraic methods can be further classified according to the specific technique used to solve the system of equations, namely as numerical, symbolic, and analysis of systems of equations.

4.3.1 Numerical Methods

Numerical methods provide a powerful tool to solve iteratively large systems of equations. In general, to guarantee convergence a good approximation of the intended solution should be supplied. This fact means that if, as it is customary, the starting point is taken from the sketch defined by the user, then the sketch should be close to the intended solution. The method offers little control over the solution in which the user is interested. To achieve robustness, numerical iterative methods must be carefully designed and implemented.

Borning, [5], Hillary and Braid, [28], and Sutherland, [82] use a relaxation method. This method is an alternative to the propagation method as we explain later on. Basically, the method perturbs the values assigned to the variables and minimizes some measure of the global error. In general, the convergence is slow.

The method most widely used is the well-known Newton-Raphson, [50]
iteration. It is used in the solvers described in [27, 64, 65, 70]. Newton-Raphson is a local method and converges much faster than relaxation. The method does not apply to consistently over constrained systems of equations unless special provisions are made such as combining it with least-squares techniques.

*Homotopy or continuation,* [2], is a family of methods with a growing popularity. These methods are global and guarantee convergence, moreover, they are exhaustive and allow to determine all solutions of a constraint problem. However, their efficiency is worse than that of Newton-Raphson. Lamure and Michelucci, [60], and Durand, [18], apply this method to geometric constraint solving.

Other, less conventional methods have also been proposed. For example, in [25], Hel-Or *et al.*, introduced the *relaxed parametric design* method where the constraints are soft, that is, they do not have to be met exactly, the problem is modeled as a static stochastic process, and the resulting system of probabilistic equations is solved using the Kalman filter familiar from control theory. The Kalman filter was developed to efficiently compute linear estimators and when applied to nonlinear systems, it does not necessarily finds a solution even if one exists.

Kin *et al.*, [51], reported on a numerical method based on extended Boltzmann machines which are a sort of neural network whose goal is to minimize a given polynomial that measures the energy of the system.

### 4.3.2 Symbolic Methods

Symbolic algebraic methods compute a Gröbner basis for the given system of equations. Algorithms to compute these bases include those by Buchberger [12], and by Wu-Ritt [14, 90]. These methods, essentially, transform the system of polynomial equations into a triangular system whose solutions are those of the given system. Then, either a forward or a backward substitution yields the solution.

Buchanan *et al.*, [11], describe a solver built on top of the Buchberger’s algorithm. In [52], Kondo reports a symbolic algebraic method. In [53], Kondo improves that work by generating a polynomial that summarizes the changes undergone by the system of equations.
4.3.3 Analysis of Systems of Equations

Methods based on the analysis of systems of equations determine whether a system is under-, well- or over-constrained from the system structure. These methods can be extended to decompose systems of equations into a set of minimal graphs which can be solved independently, [69, 80]. They can be used as a pre-processing phase for any other method, reducing the number of variables and equations that must be solved simultaneously.

Serrano, [75], applies analysis of systems of equations to select from a set of candidate constraints a well constrained, solvable subsets of equations.

4.4 Theorem Proving

Solving a geometric constraint problem can be seen as automatically proving a geometric theorem. However, automatic geometric theorem proving requires more general techniques and, therefore, methods which are much more complex than those required by geometric constraint solving.

Wu, [89, 90], applies the Wu-Ritt method, an algebraic-based geometric constraint solving method, to prove geometric theorems. The method accounts for the necessary conditions to figure out non-degenerated solutions.

In [14], Chou applies Wu's method to prove novel geometric theorems. Chou et al., in [15, 16], report on a work in automatic geometric theorem proving which allows to interpret, from a geometric point of view, the proof generated by computation.

5 The Constraint Graph

As has been said in Section 4, in the graph-based approach, solvers initially translate the problem into a graph which is then analyzed. Next we detail the graph structure and the role it plays.

5.1 The graph

In the constraint graph, the vertex set corresponds to the geometric elements of the problem. Each vertex is attributed with a weight that is its degree of freedom, normally the number of independent coordinates needed to situate
the geometric element represented. In the case of points and unconstrained lines in the plane this would be 2, in the case of circles (with no prescribed radius) it would be 3. If a line is constrained to be horizontal (or vertical), the weight would be 1.

The geometric constraints are represented by graph edges. They include dimensional constraints (angle, distance, etc.), and logical constraints (incidence, concentricity, etc.). It is customary that the sketcher infers incidence (and sometimes additional) constraints. They are also represented in the graph. Furthermore, certain transformations may be applied that change the constraint problem internally. For example, a circle with prescribed radius may be replaced with its center point after suitably changing constraints on the circle to equivalent constraints on the center of the circle. Edges are annotated by the number of degrees of freedom they remove, usually the number of independent equations. For instance, a distance constraint between two points would remove 1 degree of freedom, but an incidence constraint between two points would remove 2.

A global analysis of whether a problem is well-constrained can be done by summing the vertex weights and subtracting the sum of the edge weights. See Laman [59]. The resulting number has been called the deficiency of the problem. For coordinate-free problems (i.e., problems where the solution is not in fixed position with respect to a global coordinate system), a well constrained 2D problem should result in a deficiency of 3; e.g., [20]. When the problem is to be situated with respect to a fixed coordinate system, then a deficiency of zero would be required.

An induced subgraph can be solved if it corresponds to a well-constrained subproblem. Finding a minimal subgraph gives us a minimally complicated problem to solve. We call such a problem a cluster core. Once a subproblem has been solved, we may enlarge the subproblem sequentially by geometric elements: If the element has weight $k$ and is constrained with respect to geometric elements in the subproblem where the edge weights sum to $k$, then we may add the new element as a sequential extension of the cluster.

By analyzing the constraint graph in this way, it can be decomposed into a set of clusters that can be solved separately. Several clusters can be combined when they overlap pairwise in a single geometric element. Here, we infer constraints between the shared elements, from the solved subproblems. The shared elements can be placed with respect to each other and form a skeleton on which to place the solved clusters. In 2-space solvers employing triangle decomposition, so combining three clusters is a natural step; [6, 20].
Clusters can also be sometimes added individually in analogy to a sequential extension.

Triangle decomposition employs only 2-element cluster cores and merges three clusters that pairwise share a geometric element. Such 2D solvers are very attractive because they are reasonably competent in CAD applications and are conceptually very simple. They also require no more than solving univariate quadratic equations. The theory of such solvers is fairly well understood with respect to intensionality; [21].

If the entire constraint graph is recursively decomposable into a single cluster, then the planning phase succeeds and a solution of the constraint problem is possible in principle. If the decomposition fails, then the solver will announce that no solution can be found. This means that there is no solution based on triangle decomposition, but it does not mean that there is no solution. Here we see the difference between a particular strategy and a fully general constraint solver. There is a known algorithm that will succeed in polynomial time decomposing any graph that corresponds to a solvable constraint problem, but it is much more complex and the subsequent solver phase also is more demanding, because there cannot be an a-priori restriction on the size of minimal cluster cores; [33, 34].

5.2 Order of Decomposition

Typically the graph can be decomposed in several ways, raising the question whether we need a canonical order. There are theorems which prove that if a constraint graph can be decomposed in one way, using triangle decomposition, then every sequence of triangle decompositions fully decomposes the constraint graph. The proof is based on the Church-Rosser property of graph reduction by fusing solvable subgraphs into single nodes; [21]. These theorems can be generalized, and the order of decomposition is not critical for determining solvability, bib:hoffmann01a [35, 36].

6 Solver Phase

When the constraint graph has been analyzed and a solution plan formulated successfully, a second phase now assigns coordinates and solves equations to do so. The basic step here is to place a pair of constrained geometric elements into a standard position and then to add sequentially to this cluster core. Six
basic cases can be distinguished when the universe of geometric elements is
restricted to points and lines (and, by extension, to fixed-radius circles). One
of these cases is indeterminate. The others may have up to four solutions.
Based on heuristics, the solver selects one of these. Typically the selected
solution has the same order type as the input sketch based on the hypothesis
that the sketch captures the intent of the user. The input sketch data is
communicated by giving the initial coordinates of each element as sketched
by the user. This strategy of selecting the order type can fail when the
user varies dimensional values. It may be that under different dimensional
parameters a different order type is needed, and that the previous sketch no
longer is a good guide of design intent.

When merging three clusters, the solution can be reduced to the cluster
core/sequential extension procedure by considering one cluster (A) the core
and deriving, from the other two clusters (B) and (C), a measured constraint
between the shared element in (A) and the (third) element shared between
(B) and (C). In this way placing the third element with respect to (A) is a
sequential extension. Once placed, rigid transformations are computed that
place all other elements of (B) and of (C) with respect to the cluster (A).

If the graph analysis determines that the problem is solvable in principle,
based on a successful graph decomposition, must the solver phase necessarily
find a solution? The answer to that is clearly no. Consider stipulating the
lengths of the three sides of a triangle to be, respectively, 1, 1, and 3. Since
the triangle inequality is violated, there cannot be a solution, even though
generically there could exist one, for different side lengths.

7 Root Selection

Even sequential constraint problems have several solutions. Which one of
the (perhaps exponentially) many solutions to select is the root-selection or
chirality problem. The selection is usually made on basis of the user sketch,
preserving where possible the order type of the solution. It is noteworthy
that the strategy is invariant under decomposition order, in the case of
triangle decomposition. It can be shown that the same triples are queried for
order type when so constructing a solution, and that congruent solutions will
be determined no matter what order of decomposition is followed [20, 21].

On initial design it is reasonable to expect that the input sketch and
the solution the user wants are of the same order type. However, as we
suggested before, this is not necessarily the case when exploring variations of a sketch obtained by varying dimensional parameters and examples of this phenomenon are easy to construct. A systematic exploration of the solutions is possible using a tree in which each interior node represents a constructed solution and the descending children are obtained from the different roots of the system of equations solved at that point, [6]. This approach is not particularly user-friendly.

Automated techniques are not of promising generality. For example, consider stipulating that the constraint solution be a non self-intersecting polygon. We can show that this requirement leads to NP-hard steps in the solver, [20]. Other approaches that are based on user-interaction include dragging a single geometric element into a different position followed by a renewed run of the solver phase. This is the approach used by DCM, a commercial constraint solver. The effect is that some of the triples involving the dragged elements now have a different order type, so that a different solution would be found. This can confuse the user when several triples should be of different order type since the intermediate stages are not necessarily intuitive or suggestive of progress.

Another variation, also requiring user interaction, is to present to the user each solution step visually with each alternative shown so that a selection can be made. This is in effect the tree exploration (on demand) with a graphical method of asking for the intended branch. Clearly, all such root selection modifications should be done on demand only, and they will require user sophistication and a conceptual understanding of the chirality problem.

8 Variable-Radius Circle

When circular elements are used without a definite radius, we must extend the graph analysis to working with vertices of weight 3. These circles have been called variable-radius. One of the uses of variable-radius circles is to construct geometrically constraint definitions of the kind where two lengths should be equal.

Including variable-radius circles impacts both the basic construction steps as well as cluster merging. The sequential construction steps are fairly simple and involve, on the solver side, classical geometry of conic sections; [72].
Cluster merging entails, as additional construction step, merging three clusters where one cluster is simply a variable-radius circle. The corresponding solver steps can be complicated. The arising algebraic equations can be simplified by spatial geometric constructions, in particular by cyclographic or Laguerre maps, [13, 30, 31]. Typically, commercial solvers implement these constructions numerically, if at all.

The geometry of solving variable-radius problems using Laguerre maps can be very elegant and leads in general to simpler algebraic systems because it is capable of differentiating among the possible solutions different orientations, thereby in effect factoring the system. It is not obvious whether an abstract algebraic equivalent can be formulated that factors the equations directly. Several papers have been published that address variable-radius circle configurations and enumerate the possible configurations.

9 SolBCN. A Constructive Logic-Based Solver

To illustrate some of the concepts described so far, we briefly describe the main features of the constructive, logic-based solver SolBCN, [79].

The solver handles bidimensional geometric configurations composed of points, segments, arcs and circles. The constraints that can be defined on these objects include distance between two points, perpendicular distance between a point and a segment, angle between two segments, incidence, perpendicularity, parallelism, tangency and concentricity.

The solver is variational, i.e., the solver processes the constraints without the need of arranging them in a predefined order. Furthermore, the solver can deal with problems with circular constraints.

The basic method for solving geometric constraints is a constructive approach and exhibits properties of both rule- and graph-constructive approaches, Bouma et al., [6]. In a first phase, the solver uses rewrite rules to build a sequence of construction steps which, in a second phase, are carried out to generate a solution by placing the geometric objects according to the dimension values and constraints. We explain how the data is represented and which rules are available.
9.1 Data Representation

All constraints mentioned can be represented by means of distance between two points, distance between a point and a straight line segment and angle between two straight line segments. We use a notation derived from Verroust, [86]. The distance constraints between points are represented by a CD set, the point-segment distance constraints are represented by a CH set, and the angle constraints between two segments are represented by a CA set. We define these sets more formally as follows.

A CD set is a set of points with mutually constrained distances. A frame of reference is attached to each CD set and the points in the set are placed with respect to this frame. When a CD set contains just two points, we will refer to it as an elementary CD set. It is worth to note that a sketch is solved when all the points in the sketch belong to the same CD set.

A CH set is a point and a segment constrained by the perpendicular distance from the point to the segment. A CA set is a pair of oriented segments which are mutually constrained in angle. In what follows, we will refer generically to the CD, CA and CH sets as constraint sets.

9.2 Rules

We classify the rules into one of the following types: creation rules, merging rules or construction rules.

Creation rules create elementary CD sets, CA sets and CH sets as an interpretation of the problem sketched by the user. The sign of the distances and angles are defined based on what the user has sketched. When a distance constraint between two points is given, a CD set is created. The position of the points in the associated frame of reference are $(0, 0)$ and $(d, 0)$. (Figure 5 left.) When an angle constraint between two directed segments is processed, a CA set is created where one of the straight segments defines the positive x axis. (Figure 5 middle.) Whenever a point, a segment and the perpendicular distance from the point to the segment are given, a CH set is created. (Figure 5 right).

Merging rules include just one rule that computes the transitive closure of the angle constraint set. When a segment belongs to two different CA sets, $ca_1$ and $ca_2$, a new CA set, $ca_3$, is created which constrains the angle of two segments, one in $ca_1$ and the other in $ca_2$, both different from the
Figure 5: From left to right: Creation of elementary CD, CA and CH sets.

shared segment. See Figure 6.

Construction rules merge CD sets, CH sets, and CA sets into larger CD sets. Merging is performed by building triangles and a few quadrilaterals. A complete description reported by Joan-Arinyo et al. can be found in [43].

9.3 Implementation

Figure 7 shows the architecture of SolBCN, [48]. The geometric constraint solving problem is split into three main components: analyzer, index selector and constructor. The analyzer symbolically determines whether the problem is or is not solvable. If it is solvable, the analyzer generates a sequence of construction steps.

The selector solves the chirality problem, (see Section 7), by selecting a specific solution instance. The selection techniques implemented are user-sketch-based, and genetic algorithms. See Luzón [66] and Joan-Arinyo et al., [42]. The constructor carries out the actual construction of the geometric object by applying the sequence of construction steps generated by the solver to the actual parameter values.

There are several reasons for this architecture. First, the nature of the computations in each step is quite different. The analyzer requires symbolic computation while the constructor only performs numerical computations. Second, determining whether the problem can be symbolically solved or not is performed in the analysis step and it does not depend on the actual parameter values nor on the geometric computations. Next, when computing solutions of different parameter values using this decoupling, only the sec-
Figure 6: Merging two CA sets.

Figure 7: SolBCN architecture.
ond step needs to be recomputed. The analysis step, computationally the most expensive part, can be skipped. Finally, given a symbolically solvable problem and a set of actual values for the parameters, the solution can be instantiated if there are not quantitative inconsistencies in the parameter values. The inconsistencies are detected while carrying out the geometric computations.

The analyzer is an expert system programmed in Prolog. We chose Prolog because of its support for the symbolic computation in the analysis phase and because it is a rapid prototyping language. The constructor is a virtual machine that just executes the sequence of construction steps generated by the analyzer.

10 Spatial Constraint Solving

In 2-space sketching applications, a major application area of constraint solving, graph-based solvers have become dominant. The underlying reason is that for the constructs common in 2-space sketching a small set of subgraphs suffices, and that the associated algebraic solution problems are rather simple. In contrast, spatial constraint solving does not appear to have a simple core that suffices for most practical applications.

The problem of spatial constraint solving is that simple subgraphs of, even up to 6 vertices, are numerous and many of them correspond to associated algebraic problems that are rather difficult. There is also the problem that no consensus has arisen of the characteristic spatial constraint problems of relevance to, say, CAD, so that there is little guidance on how to select a subset from the subgraph patterns to arrive at a compact solver that is widely applicable. In this section we review some of these issues in the context of a graph decomposition solver architecture. We begin with the problem of solving the equations associated with a selection of spatial constraint problems.

10.1 Sequential Construction Problems

The simplest 3-space constraint problems require placing a single geometric element (point, plane or line) with respect to a set of geometric elements whose position and orientation are known. We call such problems sequential, since the elements are placed one-by-one sequentially.
Many, but not all, sequential problems are easy to solve. For example, placing a point with respect to three known points requires the intersection of three spheres. Elementary algebraic manipulation reduces this task to solving a univariate quadratic equation. A difficult sequential problem is placing a line such that it is at prescribed distance from four known points in 3-space. Geometrically, this is equivalent to finding common tangents to four given spheres. A lower bound was established by Macdonald et al., [67], who exhibited four spheres of equal radius that have 12 distinct, common tangents. That this bound is sharp was proved by Hoffmann and Yuan in [29] by deriving a nonlinear system of equations whose solutions give all common tangents. It was shown that the geometric degree of the system is 12, thus establishing the missing upper bound. Note that solving the system is not trivial.

10.2 Algebraic and Geometric Solutions

In the algebraic approach to solving specific constraint problems, one formulates a system of algebraic equations. The system is then simplified using algebraic manipulation and geometric reasoning, and if the resulting system is simple enough, its solutions can be computed with high reliability and accuracy. For spatial constraint problems, it is not necessarily easy to find such simple systems, and so computing solutions may require sophisticated algorithms, e.g., [18]. As we have seen, sequential problems may already require solving algebraic systems of degree 12.

A geometric approach to finding the solutions of a nonlinear equation system is the locus method, [23]. Instead of relying only on root finding techniques, a geometric idea is introduced. Drop one constraint from the problem, say a dimensional constraint \( c \), resulting in an underconstrained problem. Evaluate the underconstrained configuration and measure the actual value of the dimension. Different configurations lead to different values, resulting in a curve that can be traced. This curve is the locus of \( c \). Intersect the locus of \( c \) with the nominal constraint value, usually a straight line. The resulting intersection configurations are solutions to the original system. The method can be extended to computing the locus of more than one cut constraint, resulting in geometric manifolds of higher dimensions. Cutting two constraints we would obtain a surface that would be intersected with two planes, cutting three constraints a spatial manifold is obtained, and so on. We will illustrate the idea for octahedral problems.
10.3 Octahedral Problems

Consider a constraint graph with six vertices and with edges arranged as shown in Figure 8. We call such a graph octahedral on account of the fact that the topology is that of the vertices and edges of a regular octahedron. When the vertices represent points or planes in 3-space, and the edges therefore a distance between two points, a distance between a point and a plane, and/or an angle between two planes, then there are two configurations that correspond to underconstrained problems, namely configurations with 5 or 6 planes, [37]. The other configurations can be solved algebraically. An elegant approach to formulating the equation system is due to Michelucci, [68] who employs the Cayley-Menger determinant as follows.

The determinant expresses the distances between the five points:

\[
\begin{vmatrix}
0 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & d_{12} & d_{13} & d_{14} & d_{15} \\
1 & d_{12} & 0 & d_{23} & d_{24} & d_{25} \\
1 & d_{13} & d_{23} & 0 & d_{34} & d_{35} \\
1 & d_{14} & d_{24} & d_{34} & 0 & d_{45} \\
1 & d_{15} & d_{25} & d_{35} & d_{45} & 0
\end{vmatrix}
\]

where the \( d_{ik} = \| p_i - p_k \|^2 \) is the squared distance between the points \( p_i \) and \( p_k \). Choosing two sets of five points in the configuration, we can express the squared distances indicated in red in Figure 9 as function of the given

\footnote{The URL is no longer valid.}

26
distances. This results in two bivariate equations of degree 4 each in the two unknown distances, indicating at most 16 distinct solutions. That the bound of 16 solutions is sharp was shown earlier by Hoffmann and Vermeer in [37]. Suitable extensions of the Cayley-Menger determinant can be used for the remaining cases comprising both points and planes.

A solution of the two quartic equations can be obtained by the locus method, for example. Each quartic equation taken separately yields a plane curve. Tracing these curves can then give initial values for an iterative numerical method that isolates the up to 16 roots of the system.

The locus method can also be used to plot directly the value of a cut constraint, and we explain the approach assuming only points in the configuration. Consider Figure 10. Select the three blue vertices and cut the red constraint. The three blue vertices can be positioned using the blue constraints only, in the $xy$-plane for example.

Each of the green vertices, together with the adjacent blue vertices, can be thought of as a rigid triangle that pivots about a blue edge, and so the green vertices move each on a circle whose plane is perpendicular to the $xy$-plane. Select one green vertex and parameterize its position on the circle on which it moves, say by $\theta$. For a given $\theta$ value, we can now compute the position of the remaining two green vertices using the black constraints. Once these vertices have been positioned, we can measure the distance $d$ between the two vertices adjacent to the cut, red constraint, so plotting a curve in the $\theta d$-plane. Intersecting the curve with the line $d = d_0$, where $d_0$ is the distance stipulated by the red constraint, we obtain all solutions to
the original problem.

10.4 Line Configurations

In the discussion of sequential problems before, we have seen that line problems in 3-space may yield algebraic equation systems of considerable complexity. In addition, there is a large number of individual problems involving only a small number of geometric elements. We illustrate the combinatorial explosion of cases involving constraint graphs with 6 or fewer vertices representing points, lines and planes.

Note that between two planes, between to points, and between a point and a plane we can have at most one constraint, namely of angle, of distance, and of distance, respectively. Hoffmann and Vermeer show in [37] that the octahedron problem is the simplest, nonsequential problem involving only points and planes. As shown in that paper, there are exactly 10 distinct configurations, two of them underconstrained.

Moreover, we can have at most a single constraint between a point and a line (distance), and a single constraint between a plane and a line (angle). However, between two lines we can have up to two constraints (distance and angle). As Gao et al. show in [23], there are two distinct nonsequential constraint problems with four lines, shown in Figure 11.

The blue lines represent two constraints, of angle and distance, and the black line represents a distance constraint. The two configurations differ in the red constraint a distance constraint in one and an angle constraint in

Figure 10: Octahedral problem. The locus method.
the other configuration.

Gao et al. in [23] also establishes that there are 17 distinct configurations with 5 geometric elements, including lines, and more than 680 configurations with 6 geometric elements. These numbers show that a solver, based on decomposing a spatial constraint problem into a (recursive) set of small subproblems must have a very large repertoire of subproblem patterns, even when allowing only up to six geometric elements. Moreover, the algebraic structure of the many configurations involving lines remain unexplored.

Acknowledgements

We wish to acknowledge our collaborators over the years, especially William Bouma, Jiazen Cai, Xiangping Chen, Ching-Shoei Chiang, Cassiano Durand, Ioannis Fudos, Xiaoshan Gao, Ku-Jin Kim, Ramanathan Kavasseri, Andrew Lomosov, Robert Paige, Meera Sitharam, Antoni Soto, Pamela Vermeer, Sebastià Vila, Weiqiang Yang, and Bo Yuan. Much of the material reviewed has been to their credit.

R. Joan-Arinyo has been suported by The Ministerio de Educación y Ciencia and by FEDER under grant TIN2004-06326-C03-01.

References


[91] Y. Yamaguchi and F. Kimura. A constraint modeling system for vari-
ational geometry. In J.U. Turner M.J. Wozny and K. Preiss, editors, 
North Holland, 1990.
Departament de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Research Reports - 2004

• LSI-04-1-R : *Automatic Generation of Polynomial Loop Invariants: Algebraic Foundations*, Rodríguez, E. and Kapur, D.
• LSI-04-2-R : *Comparison of Methods to Predict Ozone Concentration*, Orozco, J.
• LSI-04-3-R : *Towards the definition of a taxonomy for the cots product’s market*, Ayala, Claudia P.
• LSI-04-4-R : *Modelling Coalition Formation over Time for Iterative Coalition Games*, Mérida-Campos, C. and Willmott, S.
• LSI-04-5-R : *Illegal Agents? Creating Wholly Independent Autonomous Entities in Online Worlds*, Willmott, S.
• LSI-04-6-R : *An Analysis Pattern for Electronic Marketplaces*, Queralt, A. and Teniente, E.
• LSI-04-7-R : *Exploring Dopamine-Mediated Reward Processing through the Analysis of EEG-Measured Gamma-Band Brain Oscillations*, Vellido, A. and El-Deredy, W.
• LSI-04-9-R : *Similarity and Dissimilarity Concepts in Machine Learning*, Orozco, J.
• LSI-04-10-R : *A Framework for the Definition of Metrics for Actor-Dependency Models*, Quer, C. and Grau, G. and Franck, X.
• LSI-04-15-R : *Evaluation and symmetrisation of alignments obtained with the Giza++ software*, Lambert, P. and Castell, N.
• LSI-04-16-R : *A note on the use of topology extensions for provoking instability in communication networks*, Blesa, M.J.
• LSI-04-17-R : *An ISO/IEC-compliant Quality Model for ER Diagrams*, Costal, D. and Franck, X.
• LSI-04-19-R : *Adding Efficient and Reliable Access Paths to the JCF*, Marco, J. and Franck, X.
• LSI-04-20-R: *Exploiting Simple Corporate Memory in Iterative Coalition Games*, Mérida-Campos, C. and Willmott, S.
• LSI-04-22-R: *Complexity issues on bounded restrictive H-coloring*, Diaz, J. and Serna, M. and Thilikos, D.M.
• LSI-04-23-R: *Chromatic number in random scaled sector graphs*, Diaz, J. and Sanwalani, V. and Serna, M. and Spirakis, P.
• LSI-04-24-R: *Bounds on the bisection width for random d-regular graphs*, Diaz, J. and Serna, M. and Wormakl, N.C.
• LSI-04-25-R: *Open Source environment to define constraints in route planning for GIS-T*, Pérez, L. and Silveira, A. da M.
• LSI-04-26-R: *A basic repository of operations for the refinement of general ontologies*, de Palol, X.
• LSI-04-27-R: *Tetrahedral mesh subdivision based on underlying volume data*, Rodríguez, L. and Navazo, I. and Vinacua, A.
• LSI-04-28-R: *The Price of Connectedness in Expansions*, Fomin, F.V. and Fraigniaud, P. and Thilikos, D.M.
• LSI-04-29-R: *Smaller kernels for hitting set problems of constant arity*, Nishimura, N. and Ragde, P. and Thilikos, D.M.
• LSI-04-30-R: *Searching Spatial Sense in the Ontological World: Discovering Spatial Objects*, Morocho, V and Pérez, L. and Saltor, F.
• LSI-04-33-R: *Multisided patches*, Pla, N. and Vigo, M. and Cotrina, J.
• LSI-04-34-R: *SVMTtool: A general POS tagger generator based on Support Vector Machines*, Giménez, J. and Márquez, LL.
• LSI-04-35-R: *A distributed and mobile component system based on the ambient calculus*, Mylonakis, N. and Orejas, F.
• LSI-04-37-R: *The AlignmentSet Toolkit*, Lambert, P.
• LSI-04-38-R: *Integración de Fuentes de Datos espaciales: análisis e implementación de una Ontología de términos espaciales: Primera Parte - Creación de una Ontología*, Ramos, Erik G.
• LSI-04-40-R: *Kernels on Structured Domains*, Valentín, L.
• LSI-04-41-R: *Determining the Structural Events that May Violate an Integrity Constraint*, Cabot, J. and Teniente, E.

• LSI-04-42-R: *Review of Statistical Word Alignment Techniques*, Lambert, P.

• LSI-04-43-R: *Algoritmos genéticos en el problema de la solución deseada Optimización de parámetros*, Barreiro, E. and Joan-Arinyo, R. and Luzón, M.V.

• LSI-04-44-R: *Generative Topographic Mapping as a constrained mixture of Student t-distributions: Theoretical developments*, Vellido, A.


• LSI-04-49-R: *A brief on constraint solving*, Hoffmann, C.M. and Joan-Arinyo, R.

Hardcopies of reports can be ordered from:

Núria Sanchez  
Departament de Llenguatges i Sistemes Informàtics  
Universitat Politècnica de Catalunya  
Campus Nord, Mòdul C6  
Jordi Girona Salgado, 1-3  
08034 Barcelona, Spain  
nurias@lsi.upc.es

See also the Departament WWW pages, http://wwwlsi.upc.es/