Summary of Basic 2D Constraint Solving[1]

Christoph M. Hoffmann

May 2004

Introduction and Scope

2D geometric constraint solving is arguable 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.

This summary reviews basic techniques that are widely available for solving 2D geometric constraint problems. It is meant as a companion to the set of slides of a constraint solving tutorial given at the ACM Solid Modeling Conference in Genova, Italy, in 2004. I restrict this summary to the basics of 2D solving. There are numerous extensions and variants of these techniques that have been published in the literature. They are recommend to the interested reader as follow-on material for study.[2]

I wish to acknowledge my collaborators over the years, especially William Bouma, Jiazen Cai, Xiangping Chen, Ching-Shoei Chiang, Cassiano Durand, Ioannis Fudos, Xiaoshan Gao, Robert Joan-Arinyo, 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.

Preliminaries

A *geometric constraint problem* consists of a set V of geometric objects (e.g., of Euclidean geometry) and a set C of constraints on them. A solution to the problem is an assignment of coordinates to the elements of V such that all constraints are satisfied, or a message to the effect that no such assignment exists. Since the problem in general is doubly exponential, solvers may be unable to find a solution even when such a solution exists. The set of solvable constraint problems can be considered a measure of the competence of the solver.

A problem is *under constrained* if there are infinitely many solutions (not congruent under rigid transformation), *well-constrained*, if there are finitely many solutions (possibly modulo a rigid transformation), and *over constrained* if the deletion of one or more constraints results in a well-constrained problem. These definitions can be made

[1] Appeared in J. Product Lifecycle Management Vol 1, pp. 143-149.
[2] See http://www.cs.purdue.edu/homes/cmh for some of the work.

precise using an algebraic formulation. Indeed, a constraint problem naturally corresponds to a set of (usually nonlinear) algebraic equations.

A good constraint solver is *competent*, in that it can solve all relevant constraint problems, has *intensionality* in that it finds solutions the user is interested in, and is *efficient* if solutions are found rapidly. For 2D solvers, there are good compromises that achieve these solver characteristics. It is more difficult to obtain solvers that exhibit persistence and stability, where *persistence* means that the same set of constraints always yields the same answer, and *stability* means that small changes of dimensional constraints result in small solution changes.

In general, solver competence is the antithesis of efficiency since constraint solving is doubly exponential. 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.

Major Approaches

Constraint solvers can be roughly classified as graph-based, logic-based, or algebraic. For 2D solvers, the graph-based approach has become dominant in CAD.

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. The solver analyzes the graph and formulates a solution strategy whereby subproblems are isolated and their solutions suitably combined. A subsequent phase then solves the subproblems and carries out the combination operations. 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.

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.

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. These techniques may include symbolic computation steps, such as Wu-Ritt or Gröbner bases techniques, as well as numerical techniques such as Newton iteration and homotopy continuation. The main advantage of algebraic solvers is the generality. In principle, an algebraic solver can be fully competent. On the down side, algebraic solvers may have low efficiency or may have difficulty constructing solutions reliably. When used to preprocess and study specific constraint systems, however, algebraic techniques can be very useful and practical.

The Constraint Graph

In graph-based solvers, the constraint problem is initially translated into a graph. The vertex set V 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 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 edge weights. The resulting number has been called the *deficiency* of the problem. For coordinate-free problems (i.e., problems where the solution can be positioned with respect to a coordinate system), a well constrained 2D problem should result in a deficiency of 3. 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 pair-wise 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 2D solvers employing triangle decomposition, so combining three clusters is a natural step. 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 pair wise 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.

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.

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 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.

Order of Decomposition

Typically the graph can be decomposed in several ways, raising the question whether we need a canonical order. There are theorems that 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. These theorems can be generalized, and the order of decomposition is not critical for determining solvability.

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.

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. 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. 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. 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 flummox the user when several triples should be of different order type as 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.

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.

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. 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.

Conclusions

In 2D, the graph-based approach to constraint solving is the dominant approach today since it achieves a good balance between competence and efficiency. Moreover, limiting to simple geometric elements, including fixed-radius circles, the triangle decomposition leads to a very simple solver phase. However, root selection remains even for such core solvers a critical issue that would benefit from algorithms with greater user friendliness.

This situation quickly changes when allowing variable-radius circles and admitting more complex graph reduction patterns. Both the graph analysis and the solver phases quickly ramp up in complexity requiring substantial algorithmic investments in both. Note, however, that the Church-Rosser property of the graph analysis remains valid.

In 3D several complications arise. The most basic geometric entities are points, planes and lines, as well as fixed-radius spheres and cylinders as the offsets of points and lines. Now even basic sequential problems may require sophisticated solvers, for example when finding a line that has prescribed distances from four given points (or, equivalently, finding the common tangents to four spheres). But also the graph analysis phase becomes troublesome: There are configurations in which the basic degrees-of-freedom counting breaks down, and there are numerous patterns to consider (leading to a large number of prototypical equation systems) even when restricting to those very basic geometric entities and small numbers of elements. Here, a compelling subset of 3D constraint patterns is not yet known, so that applications typically restrict to coupled 2D problems.