Witness computation for solving geometric constraint systems

Arnaud Kubicki [a], Sebti Foufou [a,b,*], Dominique Michelucci [a]

[a] *Le2i Lab, CNRS UMR 5158 , University of Burgundy, BP 47870, 21078 Dijon, France.*

[b] *Computer Science, College of Engineering, Qatar University, Qatar.*

Abstract

In geometric constraint solving, the constraints are represented with an equation system $F(U, X) = 0$, where X denotes the unknowns and U denotes a set of parameters. The target solution for X is noted X_T. A witness is a couple (U_W, X_W) such that $F(U_W, X_W) = 0$. The witness is not the target solution, but they share the same combinatorial features, even when the witness and the target lie on two distinct connected components of the solution set of $F(U, X) = 0$. Thus a witness enables the qualitative study of the system: the detection of over- and under-constrained systems, the decomposition into irreducible subsystems, the computation of subsystems boundaries.

This paper investigates the witness computation in various configurations. The witness computation will be studied under several numerical methods: Newton iterations from random seeds either in $\mathbb{R}$ and $\mathbb{C}$, the Broyden-Fletcher-Goldfarb-Shanno method, the Nelder-Mead simplex method. The robustness and performances of these methods will be analyzed and compared. The paper also presents the numerical probabilistic method from which the witness method was originated, and shows how the witness can be used for detecting dependent parameters within systems of geometric constraints.

Key words: Geometric constraint solving, Witness computation, Numerical algorithms

1. Introduction

In the context of Computer-Aided Design, the resolution of geometric constraint systems aims at yielding figures which satisfy spatial relationships between geometric entities. These spatial relationships may result from geometric processes such as reverse engineering, or given by the user. Instead of explicitly giving the coordinates of the points, lines, planes and other geometric entities, the user draws a sketch on which he provides, with specific tools, geometric constraints (*e.g.* distances, angles, incidences, *etc.*) between the entities. A configuration is a geometric model composed of the constrained entities or a subset of them. Solutions are given as the coordinates of the geometric entities, *i.e.* the configuration, which satisfies all the constraints.

A geometric constraint system thus consists of a 3-tuple $\mathcal{S} = (C, X, U)$ with C the set of constraints, X the set of unknowns (geometric entities), and U the set of parameters (metric values of the constraints and coordinates specified by the user). When considering equation-based solvers, one may also consider a geometric constraint system as a set of equations F and the resolution process as the search of a root for $F(U, X) = 0$.

A geometric constraint system can be under-constrained (there is an infinity of solutions, because there are too few constraints), over-constrained (there are no solutions because of some direct – structural – or indirect dependences, as well as inconsistencies) or well-constrained (there is a finite non-zero number of solutions). A system is said consistently over-constrained when it is generically over-constrained but the values of the parameters are such that there are actual solutions. Notice that a rigid system is under-constrained since its solutions may be translated and/or rotated without violating the constraints: for that reason, most methods consider well-constrainedness *modulo* rigid motions. For more formal definitions of geometric constraint systems and the levels of constrainedness, the reader may refer to [11].

* Corresponding author

Email addresses: arnaud.kubicki@u-bourgogne.fr (Arnaud Kubicki), sfoufou@qu.edu.qa (Sebti Foufou), dmichel@u-bourgogne.fr (Dominique Michelucci).

Geometric constraint solving has been an active research area in the recent decades [2,6]. A review of the basic techniques that are widely available for solving 2D and 3D geometric constraint problems can be found in [7]. Also, a review of decomposition techniques can be found in [9].

To accommodate with, and take into consideration, the specificities of geometric constraint systems, the available resolution algorithms, also called solvers, vary in many aspects such as:

- the steps of resolutions (dependence detection, correction, decomposition, resolution of subsystems, *etc.*),
- the underlying resolutions techniques (classical methods from numerical analysis, *e.g.* Newton-Raphson, graph techniques, probabilistic methods, *etc.*),
- the interaction with the user during the resolution (autonomous algorithms, semi-autonomous),
- the solution set provided to the user (a single configuration, several or all configurations).

Detecting direct and indirect dependences in systems of geometric constraints is a difficult task in which a lot of algorithms fail. Particularly, graph-based solvers fail to find non-structural dependences due to geometric theorems. The witness configuration method proposed in [13] efficiently overcomes the intrinsic limitations of graph-based methods in detecting all dependencies between geometric constraints (Section 2 defines the concept of witness for a system of geometric constraints). The computation of a basis of the vector space of the free infinitesimal motions of a typical witness is presented in [15], as well as the use of this basis to interrogate the witness for detecting all dependencies. Several other important problems of geometric constraint solving, such as the detection of maximal well-constrained subsystems or the computation of a well-constrained basis of a consistently over-constrained system are addressed in [16] using the witness configuration method.

The most important drawback of the witness configuration method is the need of a typical witness, *i.e.* a solution to a geometric constraint system, with the same sets C and X but different values of the parameters. The witness must be typical, *i.e.* have the same combinatorial properties as the solutions. Most of the time, the sketch drawn by the user is a witness, but when the system contains many incidence constraints (which must be satisfied by the witness) or when the user unknowingly puts the geometric entities of the sketch in singular positions, one may need to compute a witness.

This paper complements the witness configuration method by comparing the results of different methods used to compute the witness: Newton's iteration from random seeds in both $\mathbb{R}$ and $\mathbb{C}$, the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method, the Nelder-Mead Simplex (NMS) method.

A complete, guaranteed, robust interval-based solver is described in [5] and used in [16]. If this solver finds no solution, it proves that no solution exists. Only a complete solver brings this kind of guarantee and this degree of robustness. However, most of the time, using a complete solver is excessive: the considered systems are most of the time very under-constrained, with some redundant over-constrained parts, so much simpler and faster methods suffice.

This paper is organized as follows: Section 2 recalls the witness definition and shows why witness computing is both important and complicated. Section 4 presents a set of methods in order to compare their performances for the witness computation. Sections 5 and 6 give respectively 2D and 3D examples for which the witness is calculated. Section 7 presents the Numerical Probabilistic Method (NPM) which constitutes the source of inspiration of the witness configuration method; then discusses whether a qualitative study is possible before computing a witness with numerical methods. Section 8 studies, through two examples, the use of the witness method in detecting dependences between parameter values. Section 10 concludes the paper, and gives some perspectives for future extensions of this work.

2. Witnesses

Let us consider the equation expression of geometric constraint systems, where the constraints are represented with a set of equations $F(U, X) = 0$, where the symbol U denotes a set of parameters with prescribed values $U_T = \rho(U)$ (T for target), and X denotes the unknowns. The searched solution for X is noted X_T. A witness is a couple (U_W, X_W) such that $F(U_W, X_W) = 0$; most of the time, U_W and U_T differ (and X_W and X_T as well), so the witness (U_W, X_W) is not the wanted solution, but it has the same combinatorial features as the (U_T, X_T), even when the witness and the target lie on two distinct connected components of the solution set of $F(U, X) = 0$. Thus a witness enables the qualitative study of the system: the detection of over- and under-constrained systems, the decomposition into irreducible subsystems, the computation of subsystems boundaries. These aspects are detailed in references [13–16].

The main advantage of the witness method, compared to combinatorial (graph-based) methods, is that the witness method is able to detect dependences between constraints [15,16], not only the simplest ones, also called structural, which are already detectable by graph-based methods, but also non-structural dependences due to some known or unknown geometric property or theorem.

2.1. *Parameters as unknowns*

One approach to build a witness is to consider both U and X as unknowns. It is the approach used in [16], where the system was solved with interval analysis. It is also the approach used in this paper.

2.2. *Discarding constraints*

Another approach is to reduce the system by discarding metric constraints and keeping only incidence constraints. This is justified by the fact that smooth modifications of the values of parameters does not modify the properties of a typical witness. The resulting system (a system of incidence constraints) is smaller and easier to solve. Incidence constraints are also called projective constraints (projective geometry considers projective properties which still hold after projections). In 2D, incidence constraints are: alignments of 3 points, cocyclicity of 4 points, tangencies, point-curve incidences. In 3D, examples of incidence constraints are: coplanarity of 4 points, cosphericity of 5 points, *etc.*

Note that incidence constraints can be formulated as equations in several ways: *e.g.* the collinearity of 3 points A, B, C in 2D can be expressed as the vanishing of the determinant

$$\begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{vmatrix} = 0$$

or as the subsystem:

$$\begin{cases} (x_A, y_A, 1) \,.\, (a, b, c) = 0 \\ (x_B, y_B, 1) \,.\, (a, b, c) = 0 \\ (x_C, y_C, 1) \,.\, (a, b, c) = 0 \\ a^2 + b^2 - 1 = 0 \end{cases}$$

where the line ABC has equation: $ax + by + c = 0$, and the vector (a, b) is normalized. Yet other formulations are possible: non-Cartesian coordinates, or coordinates-free formulations (see below). Whatever formulation is chosen, discarding equations involving parameters is the simplest way to obtain a system with only projective constraints. This approach makes sense, since systems of incidence constraints can indeed be over-constrained (which can be counter-intuitive, cf. section 5.3) and a redundant over-constrainedness will be detected when studying a witness. For a system with a contradicting over-constrained subsystem, there is no solution, thus no witness: a complete solver (for instance an interval-based solver) is needed for such cases.

Discarding equations involving parameters may overshoot, since a combination of metric constraints (*i.e.* equations involving parameters) sometimes implies projective constraints. The following two examples illustrate this fact in 2D and in 3D. In 2D, if 6 points M_i are constrained by: $M_iF_1^2 + M_iF_2^2 = d^2$ (or $M_iF_1^2 - M_iF_2^2 = d^2$), then they lie on a common ellipse (or hyperbola), which 6 points generally do not. Thus discarding all equations $M_iF_1^2 + M_iF_2^2 = d^2$ yields a system which is projectively less constrained than the initial system.
In 3D, the collinearity of 3 points A, B, C can also be specified with the vanishing of the Cayley-Menger determinant [12]:

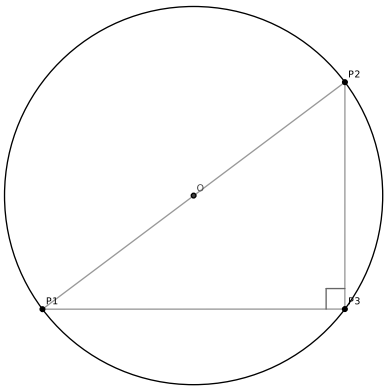

Fig. 1. A simple example : two points on a diameter of a circle and a point on the circle

$$\begin{vmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & D_{AB} & D_{AC} \\ 1 & D_{AB} & 0 & D_{BC} \\ 1 & D_{AC} & D_{BC} & 0 \end{vmatrix} = 0$$

where D_{AB}, D_{AC}, D_{BC} are squares of distances AB, AC, BC. Clearly, discarding this constraint (if some distances are parameters, and the rest represents unknowns) also cancels the incidence constraint.

Finally, apart from incidences, two numerical invariants hold in projective geometry, in 2D: the ratio of signed length between 4 collinear points (the well known cross-ratio, this stands also for the complex cross-ratio between four coplanar real points), and the ratio of signed areas of oriented triangles. Canceling equations involving these parameters is another way to form a system which is projectively less constrained than the initial system. Due to the complexity of this approach, we prefer the first one.

2.3. *Interrogating a witness : an example*

To give an example of witness interrogation, let us consider the following system of five equations:

$$0 = x_1^2 + y_1^2 - d^2 \quad (1)$$
$$0 = x_2^2 + y_2^2 - d^2 \quad (2)$$
$$0 = x_3^2 + y_3^2 - d^2 \quad (3)$$
$$0 = x_1 y_2 - y_1 x_2 \quad (4)$$
$$0 = (x_1 - x_3)(x_2 - x_3) + (y_1 - y_3)(y_2 - y_3) \quad (5)$$

The first three equations (1), (2), (3) mean that points $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ and $P_3 = (x_3, y_3)$ lie on the circle with center $(0, 0)$ and radius d. d is a parameter, which is considered as an unknown, like x_i, y_i, to compute a witness. Eq. (4) means P_1 and P_2 lie on the same diameter. Eq.

```
unknown x1 -5 5 ;
unknown y1 -5 5 ;
unknown x2 -5 5 ;
unknown y2 -5 5 ;
unknown x3 -5 5 ;
unknown y3 -5 5 ;
# parameter d is considered as unknown:
unknown d  0  20 ;
# P1, P2, P3 are on the circle of center 0 and radius d:
x1*x1 + y1*y1 - d*d = 0 ;
x2*x2 + y2*y2 - d*d = 0 ;
x3*x3 + y3*y3 - d*d = 0 ;
# P1 and P2 are on the same diameter (using determinant):
x1*y2 - y1*x2 = 0 ;
# P1P3 and P2P3 are orthogonal:
(x1-x3)*(x2-x3) + (y1-y3)*(y2-y3) = 0 ;
```

Fig. 2. The data file for the simple 2D example in Fig.1.

(5) means that P_1P_3 and P_2P_3 are orthogonal; the latter constraint is redundant, due to a geometric theorem.

One can easily show that points $P_1 = (-4, -3)$, $P_2 = (4, 3)$ and $P_3 = (4, -3)$ form a witness of this system. It is shown on Fig.1. The jacobian matrix at the witness is:

$$\begin{pmatrix} -8 & -6 & 0 & 0 & 0 & 0 & -10 \\ 0 & 0 & 8 & 6 & 0 & 0 & -10 \\ 0 & 0 & 0 & 0 & 8 & -6 & -10 \\ 3 & -4 & 3 & -4 & 0 & 0 & 0 \\ 0 & 6 & -8 & 0 & 8 & -6 & 0 \end{pmatrix}$$

and it has rank four, showing that one equation is a consequence of the others. Remark that for a random point, the rank is maximum (*i.e.* five here).

3. Implementation issues

We choose to represent the constraints as a systems of equations and inequalities. It may also be represented in a more abstract way before translation into equation form. See Fig.3 the data file for the problem of two rigid triangles discussed in section 5.1.

Expressions in equations and inequalities are represented as DAG (Directed Acyclic Graph), roughly trees with sharable nodes. A DAG is either a given number, a variable given by its name (a parameter or an unknown), a binary operation $(+, \times, -)$, a power (eg x^3), a trigonometric function (cos). Many operations can be performed on DAGs: symbolic operations (symbolic derivation, simplification, pretty printing, compilation, static analysis) and evaluation operations (floating-point arithmetic, evaluations with interval arithmetics, substitution –a kind of symbolic evaluation–, conversion to polynomials when possible, etc). DAG evaluation uses the Visitor design pattern [18] to ease extension and evaluation with other mathematical objects (dual numbers for example). Many implementations (recursive, or non-recursive, hash-consing, some kind of compilation or optimization) were investigated, because DAG evaluations may be a complex operation.

```
unknown x1  -30.000000 30.000000 ;
unknown y1  -30.000000 30.000000 ;
unknown x2  -30.000000 30.000000 ;
unknown y2  -30.000000 30.000000 ;
unknown x3  -30.000000 30.000000 ;
unknown y3  -30.000000 30.000000 ;
unknown x4  -30.000000 30.000000 ;
unknown y4  -30.000000 30.000000 ;
unknown x5  -30.000000 30.000000 ;
unknown y5  -30.000000 30.000000 ;
unknown x6  -30.000000 30.000000 ;
unknown y6  -30.000000 30.000000 ;
unknown d12  0.000000 100.000000 ;
unknown d23  0.000000 100.000000 ;
unknown d31  0.000000 100.000000 ;
unknown d45  0.000000 100.000000 ;
unknown d56  0.000000 100.000000 ;
unknown d64  0.000000 100.000000 ;
unknown d14  0.000000 100.000000 ;
unknown d25  0.000000 100.000000 ;
unknown d36  0.000000 100.000000 ;
0= (x1  - x2 )^2  + (y1  - y2 )^2  - d12^2 ;
0= (x2  - x3 )^2  + (y2  - y3 )^2  - d23^2 ;
0= (x3  - x1 )^2  + (y3  - y1 )^2  - d31^2 ;
0= (x4  - x5 )^2  + (y4  - y5 )^2  - d45^2 ;
0= (x5  - x6 )^2  + (y5  - y6 )^2  - d56^2 ;
0= (x6  - x4 )^2  + (y6  - y4 )^2  - d64^2 ;
0= (x1  - x4 )^2  + (y1  - y4 )^2  - d14^2 ;
0= (x2  - x5 )^2  + (y2  - y5 )^2  - d25^2 ;
0= (x3  - x6 )^2  + (y3  - y6 )^2  - d36^2 ;
```

Fig. 3. The data file for the two rigid triangles.

4. Methods

Four different methods have been experimented in order to generate a witness: Newton iterations in $\mathbb{R}$, Newton iterations in $\mathbb{C}$, Nelder-Mead downhill simplex method [17], and the BFGS method. Random points are used as starting seeds for all of these methods.

To avoid long time processing in our experiments, we have limited the number of iterations to 100 iterations, beyond this number we consider that the method fails to converge.

All the tried methods converge if the starting point is a root, or is close enough to a root. Proximity of starting points to a root helps the solver to converge to this root, but this convergence depends on the underlying methods of the solver. The basins of attraction of a root depend on the used solver; for instance Newton basins are fractals, homotopy basins have smooth boundaries.

4.1. *The Newton method*

Given a good starting point, the main advantage of this method is the high rate of convergence (superquadratic).

This method is also easy to implement in its principles, we compute the symbolic derivatives of the function (this is computed once). In order to use the Newton method when the Jacobian matrix is not invertible or not square, a pseudo-inverse is calculated using Singular Value Decomposition. Let $N : X \rightarrow = X - J^{-1}F(X)$ be the Newton iteration where J^{-1} denotes either the inverse or the pseudo inverse of the Jacobian matrix of F at X. The Newton iteration is then defined as follow :

$$X_{n+1} = N(X_n)$$

X_0 being the starting point.

4.2. *The Nelder-Mead downhill simplex method*

The Nelder-Mead downhill simplex method (NMS) [17] is a minimization technique in two steps: the minimization of $\|F\|$, and a check for $\|F\|$ to be zero (to avoid local minima).

This method has the advantage of avoiding the computation of the derivative, but it is not efficient in term of number of function evaluations (due to the high number of iterations). Given a big starting simplex (computed from the initial random starting point), the algorithm computes the minimum of the function with different kinds of steps such as:

- the computation of the maximum of the function on the simplex and then the "reflection" of the vertex towards the opposite hyper-face, or
- the use of the previous reflection step plus an expansion of the simplex, or
- the contraction of the simplex toward the highest point *i.e.* the vertex which maximizes $\|F\|$, or the contraction in multiple directions toward the lowest point *i.e.* the vertex which minimizes $\|F\|$.

These different operations may be combined in a way to make the simplex "walk" towards a minimum. A more in-depth description of the algorithm is available in [19].

4.3. *The BFGS method*

This is also a minimization method, hence a procedure similar to the one of the NMS method is applied. This method is a variant of the Newton method, and requires the computation of the gradient. It constructs an approximation of the Hessian matrix which is guaranteed to be definite positive in order to be sure that the algorithm makes $\|F\|$ decrease [19].This approximation is then updated with an iterative process and used to compute the next step.

4.4. *The interval Newton method*

When a complete solver is needed, we use the Newton method with intervals.

method	success rate	total time
Newton	100%	0.24 s
Newton in $\mathbb{C}$	100 %	0.26 s
BFGS	100 %	3.88 s
NMS	0 %	3.88 s

Table 1
Computing the witness of the two rigid triangles

Here, $[X]$ denotes an interval vector (*i.e.* a box) and X a vector. X_c denotes the center of the box. The center evaluation of a function G for the box $[X]$ is defined as follows:

$$G([X]) \in G(X_c) + G'([X])([X] - X_c)$$

We define the interval Newton iteration N_I by taking $G = N$ (cf. section 4.1). Note that if we use the naive interval evaluation, we can never contract the interval because when we add two intervals, the result is always larger. As the roots are both in $[X_n]$ and $[X_{n+1}]$ we then compute $[X_{n+1}]$ as $(N_I([X_n])) \cap [X_n]$. When we can't reduce the studied interval, we bisect it and study each half separately.

Two optimizations are possible:

- the Segupta-Hansel optimization consists in taking the most up-to-date interval when updating X.
- take into account, and study, the monotonicity of F (for example if $[X] = [a, b]$ and f is increasing then $f[X] = [f(a), f(b)]$)

We have used only the first optimization.

5. Results in 2D

In this section, we give several 2D examples of geometric constraint systems and the respective results of the different methods mentioned in Section 4.

The computer we use for the experiments is a quad core Xeon with about 4 GB of RAM.

The complete solver described in section 4.4 needs more time (several minutes) than other methods to find the solutions. To conduct our experiments in a practical way, the running time is limited to 10 minutes, beyond which we consider that the solver doesn't converge.

5.1. *Two rigid triangles*

This 2D example is composed of two triangles ABC and $A'B'C'$; each triangle is constrained with the lengths of its three edges, see Fig. 4. The assembly of the two triangles is rigidified with 3 constrained distances: AA', BB', CC'. It is sufficient to pick at random six points in 2D to get a typical witness.

Except NMS, the tried methods succeeded to find a witness as shown by the Table 1.

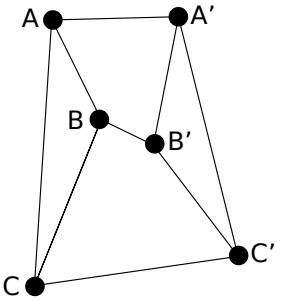

Fig. 4. A rigid 2D system made of two rigid triangles linked by three distance constraints

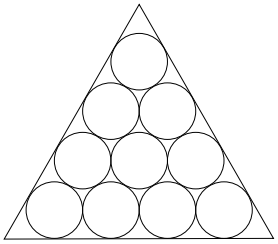

Fig. 5. An example of circle packing. For simplicity the triangle is equilateral and all circles have the same radius.

5.2. *Circle packing*

Circle packing is a 2D configuration of circles tangent to each others, see Fig. 5 for an example. Circle packing is useful in approximating some analytical functions, in graph embedding, and random walks [21]. Any planar map, or simplicial 2D complex, can be represented with a continuum of circle packing: each vertex is represented as a circle, and each edge AB in the planar map, or the simplicial 2D complex, means that the two circles for vertices A and B are outwardly tangent.

There are a lot of degrees of freedom for the shape of the exterior contour of the circle packing. For instance, it is possible to find a circle packing where all exterior circles are tangent to a common circumscribing circle. This property is a discrete analogue to Riemann theorem: there is a conformal map between any topological (open) disk and the unit disk. Some specific algorithms [3] converge iteratively to the circle packing of any given simplicial 2D complexes; Those algorithms work in interactive time for thousands of circles; they are much faster than general purpose numerical solvers.

For simplicity, we fill triangles with circles, as in Fig. 5. Let k be the number of circles tangent to one side of the triangle. We generated the systems of geometric constraints for $k = 1, 2, 3, 4, 5$, so there are 1, 3, 6, 10, 15 circles totally. The parameters are the coordinates of the three vertices of the triangle; when computing a witness, they are unknowns, as are the radii and the coordinates of the centers of circles. We do not check if the computed circles indeed lie inside the triangle, nor that they are outwardly tangent since this is not needed for a witness.

For this example, among the standard numerical methods, the best results are given by the Newton algorithm. Remark in passing that for this example it does not make sense to discard equations involving distance parameters.

k	method	success rate	total time
1	Newton	98%	0.42 s
	Newton in $\mathbb{C}$	100 %	0.72 s
	BFGS	5 %	6.08 s
	NMS	0 %	1.82 s
2	Newton	68%	1.34 s
	Newton in $\mathbb{C}$	82 %	1.42 s
	BFGS	0 %	14.7 s
	NMS	0 %	4.88 s
3	Newton	36%	3.3 s
	Newton in $\mathbb{C}$	86 %	2.5 s
	BFGS	0 %	25.88 s
	NMS	0 %	10.56 s
4	Newton	14%	6.46 s
	Newton in $\mathbb{C}$	78 %	5.06 s
	BFGS	5 %	45.62 s
	NMS	0 %	24.24 s
5	Newton	4%	11.46 s
	Newton in $\mathbb{C}$	82 %	8.9 s
	BFGS	0 %	70.84 s
	NMS	0 %	47.96 s
6	Newton	4%	32.28 s
	Newton in $\mathbb{C}$	83 %	25.96 s
	BFGS	0 %	162.24 s
	NMS	0 %	137.5 s
7	Newton	0%	50.44 s
	Newton in $\mathbb{C}$	81 %	42.44 s
	BFGS	0 %	229.72 s
	NMS	0 %	225.76 s
8	Newton	0%	78.74 s
	Newton in $\mathbb{C}$	83 %	67.9 s
	BFGS	0 %	316.4 s
	NMS	0 %	352.08 s

Table 2
Circle packing (witness)

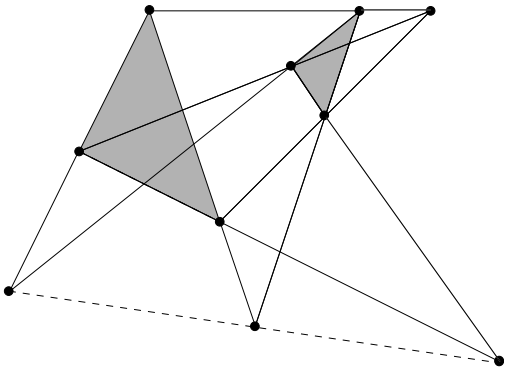

Fig. 6. An example of Desargues configuration.

method	success rate	total time
Newton	91%	6.28 s
Newton in $\mathbb{C}$	97 %	6.28 s
BFGS	0 %	24.56 s
NMS	0 %	16.78 s

Table 3
Desargues configuration (with lines)

method	success rate	total time
Newton	100%	0.54 s
Newton in $\mathbb{C}$	100 %	0.78 s
BFGS	100 %	12.08 s
NMS	100 %	1 s

Table 4
Desargues configuration (vanishing of determinants)

method	success rate	total time
Newton	100%	1.94 s
Newton in $\mathbb{C}$	100 %	2.66 s
BFGS	17 %	6.62 s
NMS	0 %	13.48 s

Table 5
Pappus configuration (lines)

5.3. *Desargues configuration*

The Desargues theorem ensures that if 2D points o, p_i, q_i are collinear, for $i = 0, 1, 2$, then the three intersection points $p_i p_{i+1 \bmod 3} \cap q_i q_{i+1 \bmod 3}$ are collinear as well, see Fig. 6. This theorem holds when points o, p_i, q_i are coplanar, and when they lie in 3D.

The objective of this example is to find points o, p_i, q_i in Desargues configuration; there are 9 alignments due to the hypothesis and one due to the conclusion. Actually, any one of the alignments results from the others. The system is both very under-constrained (no distance and no angle is specified), and also consistently over-constrained.

We have two possible formulations for alignments:

(i) use the vanishing of the determinant, or
(ii) define each line with a triple (a, b, c) with $a^2 + b^2 = 1$, and a point (x, y) lies on the line iff $ax + by + c = 0$.

Tables 3 and 4 shows the percentage success and computation time of each method. In general, Newton method performs better than the others. Results are better when the determinants formulation is used than when the lines formulation is used. In this case, all the methods are able to find a witness in less than 1 second as shown in Table 4.

5.4. *Pappus configuration*

Pappus theorem ensures that if points p_1, p_2, p_3 are collinear, and points q_1, q_2, q_3 are collinear, then the 3 intersection points $r_{ij} = p_i q_j \cap p_j q_i, i \neq j$ are collinear as well (see Fig. 7).

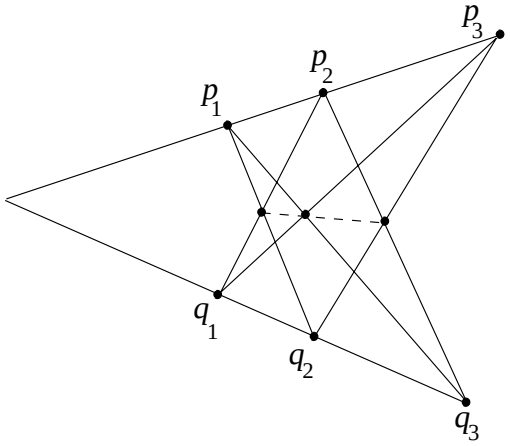

Fig. 7. An example of Pappus configuration.

method	success rate	total time
Newton	100%	0.26 s
Newton in $\mathbb{C}$	100 %	0.2 s
BFGS	100 %	3.14 s
NMS	0 %	3.36 s

Table 6
Pappus configuration (determinants)

The objective of this example is to find points p_i, q_j, r_{ij} in Pappus configuration; the nine alignments are specified: eight are due to the hypothesis, and the ninth is the conclusion of Pappus theorem. The system is both very under-constrained (no distance and no angle is specified), and also consistently over-constrained: one alignment is a consequence of the other 8 alignments; actually, due to symmetry, any one of the nine specified alignments is a consequence of the others.

We consider again the two formulations for alignment given in section 5.3. Except NMS, all the numerical methods converged with both formulations, and the best results are given by the Newton algorithm. Notice also that computation time is lower when formulation (ii) is used, see Tables 5 and 6).

5.5. *Pascal configuration*

Pascal theorem ensures that if six points lie on a common conic, then opposite sides of the hexagon (for any permutation of the vertices) intersect in three collinear intersection points. Another formulation is as follows: if a 2D hexagon (possibly self-intersecting or concave) $p_0 p_1 p_2 p_3 p_4 p_5$ is such that its three opposite sides meet in three collinear points $p_0 p_1 \cap p_3 p_4$, $p_1 p_2 \cap p_4 p_5$ and $p_2 p_3 \cap p_5 p_6$, then the same property holds for all permutations of $p_0 p_1 p_2 p_3 p_4 p_5$, see Fig. 8.

In our tests, we considered the hexagon $p_1 p_0 p_2 p_3 p_4 p_5$. We generated the system with all constraints of the hypothesis

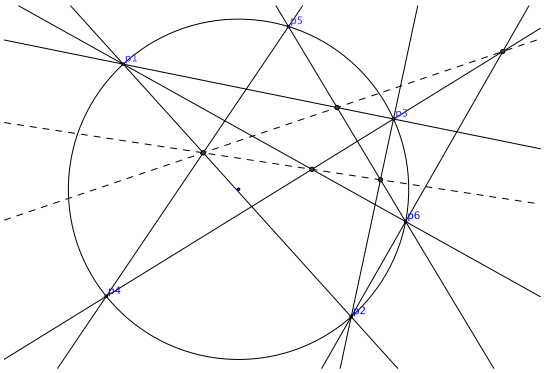

Fig. 8. An example of Pascal configuration.

method	success rate	total time
Newton	100%	0.24 s
Newton in $\mathbb{C}$	100 %	0.3 s
BFGS	100 %	4.72 s
NMS	0 %	4.46 s

Table 7
Pascal configuration

of the theorem, and the constraint of the conclusion. The alignments of three points were expressed with the vanishing of the 3×3 determinant of (homogenized) coordinates of the three points. The system was consistently redundant.

Except NMS, all the methods found a witness. The quickest is the Newton algorithm, see Table 7.

6. Results in 3D

In this section, we give 3D examples of geometric constraint systems with the results of the methods we have implemented and tested.

6.1. *Desargues configuration*

The 3D Desargues configuration is similar to the 2D one: the 3D points o, p_i, q_i are collinear, for $i = 0, 1, 2$, but the plane $p_0p_1p_2$ and the plane $q_0q_1q_2$ are different; in the generic case, these two planes meet along a line, which contains the three intersection points $p_ip_{i+1 \bmod 3} \cap q_iq_{i+1 \bmod 3}$ between homologous sides, thus these three intersection points are collinear. It is possible to formulate the theorem in a more general way, to account for degenerate cases (the planes p_i and q_i are parallel or equal, or one line $p_ip_{i+1 \bmod 3}$ is parallel to its homologous $q_iq_{i+1 \bmod 3}$, etc.), but we prefer to discard them for simplicity.

We have generated the constraints of alignment for the hypothesis and for the conclusion of the theorem. All these constraints are projective, *i.e.* they are all incidence constraints: so there is no parameter of distance or angle.

method	success rate	total time
Newton	100%	3.14 s
Newton in $\mathbb{C}$	100 %	3.44 s
BFGS	100 %	18.72 s
NMS	100 %	1.02 s

Table 8
Desargues 3D configuration

We formulate the collinearity of three points with determinants in two ways. In the first formulation, we use the vanishing of two determinants:

$$\begin{vmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{vmatrix} = \begin{vmatrix} x_1 - x_0 & x_2 - x_0 \\ z_1 - z_0 & z_2 - z_0 \end{vmatrix} = 0$$

In the second formulation we use also:

$$\begin{vmatrix} y_1 - y_0 & y_2 - y_0 \\ z_1 - z_0 & z_2 - z_0 \end{vmatrix} = 0$$

This last formulation is redundant for each alignment. All methods converge (at least on all random values tested) the quickest is the NMS, see Table 8.

6.2. *Beltrami configuration*

Let $W_i, i = 0, 1, 2$ be three non-coplanar white lines in 3D, and let $B_j, j = 0, 1, 2$ be three non-coplanar black lines in 3D, such that all pairs of a white line W_i and a black line B_j are coplanar (they meet at a point Q_{ij}, possibly at infinity). A line is called black (white) if it intersects the three white (black) lines. Then all black lines are coplanar to all white lines. Coxeter [4] credits Beltrami for this 3D incidence theorem. One can note that the white and black lines generate a ruled quadric surface, for instance an hyperboloid of one sheet, or an elliptic hyperboloid.

We reformulate the theorem as follows: Given a set of 16 points in 3D: P_{ij}, $i = 0, 1, 2, 3$ and $j = 0, 1, 2, 3$ such that for all i, P_{i0}, P_{i1}, P_{i2} are collinear, P_{i0}, P_{i1}, P_{i3} are collinear, and symmetrically for all j, P_{0j}, P_{1j}, P_{2j} are collinear, P_{0j}, P_{1j}, P_{3j} are collinear. Any of these 16 alignments is a consequence of the 15 others.

For a human being finding a witness is not trivial in this case: it is not sufficient to pick 16 points at random.

We have used two different formulations to generate these 16 constraints of alignment. In a first formulation, the alignment of 3 points is expressed as the vanishing of two determinants. In a second formulation, each alignment is expressed as the vanishing of three determinants, which is redundant. Here the Newton algorithm has always converged for both formulation, see Tables 9 and 10.

method	success rate	total time
Newton	100%	1.72 s
Newton in $\mathbb{C}$	100 %	1.72 s
BFGS	100 %	26.56 s
NMS	100 %	1.32 s

Table 9
Beltrami configuration (2 determinants)

method	success rate	total time
Newton	100%	7.22 s
Newton in $\mathbb{C}$	100 %	7.76 s
BFGS	100 %	44.3 s
NMS	100 %	1.96 s

Table 10
Beltrami configuration (3 determinants)

method	success rate	total rate
Newton	99%	0.74 s
Newton in $\mathbb{C}$	100 %	0.48 s
BFGS	48 %	15.62 s
NMS	0 %	7.12 s

Table 11
Line tangent to four given spheres

method	success rate	total time
Newton	100%	0.68 s
Newton in $\mathbb{C}$	100 %	0.7 s
BFGS	30 %	9.88 s
NMS	0 %	11.16 s

Table 12
Octahedron

6.3. *Line tangent to four given spheres*

This underlying 3D problem was first considered and solved by Hoffmann and Yuan in [8]: compute the lines tangent to 4 given spheres.

To generate a witness is trivial (at least for a human being): generate randomly a line and 4 centers for the spheres; then deduce the 4 radii: they are the distances to the line of the 4 centers.

We generated the related system of geometric constraints. The line is represented by the vanishing of 2 determinants.

Here the best results were given by Newton method with complex numbers, see Table 11.

6.4. *Octahedron*

3D polytopes (convex polyhedrons) are completely described with the lengths of their edges, and the coplanarities of vertices lying on a common face. In other words the related systems are rigid: it is Cauchy theorem.

A specific algorithm computes a realization (consistent coordinates for vertices) of 3D polytopes, given their graph (called skeleton, or 1-skeleton); it computes a 2D Tutte embedding of the graph of the polytope and lifts it in 3D [20]. Remarkably, all coordinates are rational numbers, and even integers after some scaling. This algorithm is a computational proof of Steinitz theorem: all convex polytopes have an integer realization whatever their topology (*i.e.* graph). This algorithm should be used to compute a witness for a polytope with a described graph. Steinitz theorem does not extend beyond 3D.

For an octahedron, all faces are triangles, so there is no coplanarity constraints; thus generating a witness is trivial: just pick 6 vertices in 3D space at random, then measure the lengths of its edges. Remark that the generated polytope may be self intersecting, or concave, but it does not matter: it is a witness.

We have generated the system of 12 distance constraints; both coordinates of the 6 vertices and the 12 distances are unknowns.

We have also tried to solve the related target system, *i.e.* we fixed the 12 distances parameters, and looked if the solvers still succeed (see Table 17 and Table 12).

6.5. *Octahedron with Cayley-Menger determinants*

The distances between five 3D points are not independent: the related Cayley-Menger determinant must vanish. Thus Cayley-Menger determinants [12] provide a simple solution to the octahedron problem, or to the Stewart platform. Write the Cayley-Menger condition for 4 "equatorial" (generally non-coplanar) vertices and the North vertex; write the Cayley-Menger condition for the 4 "equatorial" vertices and the South vertex. These 2 equations involve 2 unknown distances, the distances between opposite vertices among the 4 "equatorial" vertices; all other distances are known by hypothesis. This system is much simpler than the naive system (initially containing 12 unknowns and equations, easily reducible to 9 when pinning one triangle on the Oxy plane). Unfortunately, we do not know how to extend this Cayley-Menger idea to other polytopes.

We generated the related system of 2 equations; for computing a witness, all distances (the 12 edge lengths and the 2 distances between pairs of opposite "equatorial" vertices) are considered as unknowns.

We have also tried to solve the related target system, *i.e.* we fixed the 12 distances parameters, and checked if the solvers still succeed (see Table 17 and Table 13).

6.6. *Hexahedron*

The cube is an hexahedron: the graph of the hexahedron is the graph of the cube. The hexahedron has 6 (coplanar)

method	success rate	total time
Newton	89%	5.9 s
Newton in $\mathbb{C}$	100 %	4.06 s
BFGS	80 %	129.16 s
NMS	0 %	54 s

Table 13
Octahedron (Cayley-Menger)

method	success rate	total time
Newton	100%	1.96 s
Newton in $\mathbb{C}$	100 %	2.16 s
BFGS	14 %	57.42 s
NMS	0 %	34.06 s

Table 14
Hexahedron

method	success rate	total time
Newton	100%	2.24 s
Newton in $\mathbb{C}$	100 %	2.34 s
BFGS	100 %	41.5 s
NMS	0 %	56.46 s

Table 15
Icosahedron

quadrilateral faces. It is described unambiguously (up to its location and orientation in 3D space) with the coplanarity conditions and the lengths of its 12 edges (at least for generic edges lengths).

Due to the coplanarity conditions, it is not sufficient to choose 8 vertices at random in 3D. However it is easy to find a witness: the Platonic cube is an obvious witness for all hexahedra. However it is not a typical witness.

We generated the related system; for the witness, the lengths of 12 edges are considered as unknowns. The coplanarity of four vertices is specified as the vanishing of the 4×4 determinant of the (homogeneous) coordinates of the four involved vertices.

We have also tried to solve the related target system, *i.e.* we fixed the length of the 12 edges, and checked if the solvers still succeed, we found out that NMS method fails, and the success rate of the BFGS is too low, see Table 14.

6.7. *Icosahedron*

An icosahedron is a regular polyhedron with 20 identical equilateral triangular faces, 30 edges and 12 vertices. There is no coplanarity condition, so generating a witness is trivial, it suffices to pick at random 12 points in 3D space; the resulting polytope may be self intersecting, or concave, but it is a witness.

We generated the related system; for the witness, all 30 edges lengths are considered as unknowns. The Newton method gives a result with a rate of 100%, see Table 15.

method	success rate	total time
Newton	100%	5.6 s
Newton in $\mathbb{C}$	100 %	5.8 s
BFGS	100 %	182.38 s
NMS	100 %	3.02 s

Table 16
Dodecahedron

6.8. *Dodecahedron*

The graph of the dodecahedron is the one of the regular, Platonic, dodecahedron. The dodecahedron is made of 12 pentagonal faces, 20 vertices, 30 edges. It is the dual of the icosahedron. Due to the coplanarity of the pentagonal faces, it is not sufficient to generate 20 vertices at random to get a witness.

We generated the related system; for the witness, all 30 edges lengths are considered as unknowns. The coplanarity of five vertices $ABCDE$ in a common face is specified with the vanishing of two 4×4 determinants, $|ABCD| = |ABCE| = 0$. These 24 coplanarity conditions are independent, see Table 16.

Table 17 summarizes our experimental configurations, both in 2D and in 3D, for witness computations and lists the methods with the best convergence rates.

7. The Numerical Probabilistic Method

Before computing a witness, is it possible to perform a qualitative study of the system (where both parameters and unknowns must be considered as unknowns)? Of course, we can not use a witness-based decomposition.

Several approaches may be considered for this qualitative study. We can use either a graph-based method like [1] to detect structural dependences, with all the limitations of this type of methods, or the Numerical Probabilistic Method (NPM). NPM is a bit more powerful than graph-based methods: NPM decides in polynomial time the rigidity of frameworks (*i.e.* all constraints are point-point distances, with generic values; there is no incidence constraint at all) in 3D or beyond, while the combinatorial characterization of rigidity in 3D and beyond is unknown.

The main idea of the NPM is to study the Jacobian of a system $F(X) = 0$ (or $F(U, X) = 0$) at a random (thus generic) value for X (or for X and U). It is easy and worthwhile to check that equations are independent at a random point (call it a weak witness for convenience). If the Jacobian does not have full rank, then with probability one, the equations of F are dependent. Indeed, assume that the equations of F are independent, except in a set of measure 0 (which is called the singular manifold), then the probability is null to randomly select a point in this set.

Clearly, we can use the NPM before computing a witness,

System	Dimension	#unknowns	#parameters	#equations	Best method	Success rate
Two rigids	2D	12	9	9	Newton	100%
Circle packing (k)	2D	$2k$	3	$\Sigma_{i=0}^{k}(3i)$	Newton in $\mathbb{C}$	82%
Desargues	2D	30	0	20	Newton	100%
Pappus	2D	45	0	36	Newton in $\mathbb{C}$	100%
Pascal	2D	22	0	11	Newton	100%
Desargues	3D	50	0	40	NMS	100%
Beltrami	3D	48	0	32	NMS	100%
Line tangent to 4 spheres	3D	28	0	10	Newton in $\mathbb{C}$	100%
Octahedron (metrics are unkwnowns)	3D	30	0	12	Newton	100%
Octahedron (metrics are parameters)	3D	18	12	12	Newton	100%
Octahedron (Cayley-Menger)	3D	14	0	2	Newton in $\mathbb{C}$	100%
Hexaedron (unknown param.)	3D	36	0	18	Newton	100%
Hexaedron (valued param.)	3D	24	12	18	Newton in $\mathbb{C}$	40%
Icosahedron (unknown param.)	3D	66	0	30	Newton	100%
Icosahedron (valued param.)	3D	36	30	30	Newton	14%
Dodecahedron (unknown param.)	3D	91	0	55	NMS	100%
Dodecahedron (valued param.)	3D	61	30	55	Newton	97%

Table 17
The experimental systems (number of unknowns, number of parameters, number of equations) and the best convergence rates of the methods we have tried to compute the witness. In the first column, "unknown param." means that parameters are considered as unknowns; "valued param." means that parameters have specified numeric values. Parameters were given in a way which ensures the existence of a solution.

to detect some dependences and to compute the smallest dependent sets with the method in [16], or to decompose the system. However, we must be conscious of the limitations of the NPM. The point where the Jacobian rank is computed is random, so it does not lie on the solution set of $F(X) = 0$ (or $F(U, X) = 0$) which means that the random point is not representative of the solutions. NPM becomes unsafe when incidence or projective constraints are used: the witness method was introduced to overcome this limitation of the NPM.

8. Detecting dependent parameters from a witness

Parameters should be independent, at least in a correct system of constraints. Such dependences do not prevent the computation of a witness. The witness method, *i.e.* studying the Jacobian of the system at the witness, permits to detect the dependences between parameters. Previous articles [13–16] did not present this feature, which is illustrated now with two simple examples.

First example. The system: $x - u = x - v = 0$, has 1 unknown x and 2 parameters u, v. A witness for this system is easily computed. The parameters u and v are clearly dependent: $u = v$. Note that if u, or v is considered as an unknown, then the system is indeed correct; for instance, let us rename v as y: the system becomes $x-u = x-y = 0$, which makes sense.

Second example. The system: $x + y = u$, $x + z = v$, $y - z = w$ has 3 unknowns x, y, z and 3 parameters u, v, w. A witness for this system is easily computed. The parameters u, v, w are dependent: $w = u - v$. Here again, when parameters are considered as unknowns, the 3 equations are independent, since the Jacobian contains the 3×3 identity submatrix when deriving relatively to u, v and w. The witness method detects the dependence when considering the Jacobian for variables x, y, z (and discarding variables u, v, w): the 3×3 submatrix has rank 2 because the last row is the difference of the first and the second.

This difference in the rank of the same set of rows, depending on whether the columns of parameters are considered or not, permits the detection of dependences between parameters. Recall that each row contains the derivative of an equation, and each column is related to a variable, an unknown or a parameter.

We mentioned that discarding constraints involving parameters may yield a system which is projectively weaker (less constrained) than the initial system. Actually this happens when parameters are not independent. Precisely, the method proposed in this section detects such dependences between parameters. We conclude this section with examples of dependences between parameters: for 4 points in 2D there are 6 distance parameters, but only 5 of them are independent. For 5 points in 3D there are 10 distance parameters, but only 9 of them are independent. The method proposed in this section allows the detection of these dependences. Similarly, it can detect dependence in systems containing not only distances but also angles.

9. Further works

This work can be completed with the following investigations. First, start from a degenerate solution (for example all points are equal) and make a random walk on the tangent space to the solution manifold: pick a random direction in the tangent space, then perform a prediction-correction

step; this method clearly applies to all homogeneous systems.

Second, investigate a damped variant of the Newton method: instead of moving forward ΔX, we only move forward $\frac{\Delta X}{10}$ or so; we also imagine to perform a line search along the Newton direction.

Finally, we will try to perform some decomposition before computing the witness (cf section 7). Once a witness is known, we plan also to test re-parametrization [10].

10. Conclusion

This paper compared several methods to compute a witness. For all these methods, starting points were random. No attempt was made to decompose the systems. Among the tested numerical methods, the Newton method is the winner in term of both rate of convergence and time performance. In most of the cases, this algorithm converges to a witness after a reasonable number of iterations. See Table 17 for an overall view of our experimental systems and the methods with the best success rates.

In some cases, the Newton method is also able to solve the related target systems. The Newton method achieves a better rate of converge in $\mathbb{C}$ but at a cost of a small overhead.

The paper also shows that it is easier to compute a witness than to solve the target system directly. It is also easier to find a witness with a standard numerical method than with a complete solver.

From this study we derive the following strategy: to compute a witness, try the Newton method starting from 50 random seeds and when failing, resort to a complete solver like an interval solver.

Acknowledgement

This research work has been funded in part by NPRP grant number NPRP $09-906-1-137$ from the Qatar National Research Fund (a member of the Qatar Foundation). The statements made herein are solely the responsibility of the authors.

References

[1] S. Ait-Aoudia, R. Jegou, and D. Michelucci. Reduction of constraint systems. In *Proceedings of the Compugraphics Conference*, pages 83–92, Alvor, Portugal, 1993.

[2] B. Brüderlin and D. Roller, editors. *Geometric Constraint Solving and Applications*. Springer, 1998.

[3] C. Collins and K. Stephenson. A circle packing algorithm. *Computational Geometry: Theory and Applications*, 25(3):233–256, 2003.

[4] H. S. M. Coxeter. *The beauty of geometry: twelve essays*. Dover Publications, 1999.

[5] C. Fünfzig, D. Michelucci, and S. Foufou. Nonlinear systems solver in floating point arithmetic using LP reduction. In *SPM '09: Proceedings of the SIAM/ACM joint conference on Geometric and Physical Modeling*, pages 123–134, San Francisco, CA, U.S.A., 2009.

[6] X.-S. Gao and D. Michelucci. Special issue on geometric constraints, guest editors' foreword. *International Journal of Computational Geometry and Applications*, 16(5–6):377–378, 2006.

[7] C. M. Hoffmann and R. Joan-Arinyo. A brief on constraint solving. *Computer-Aided Design and Applications*, 2(5):655–663, 2005.

[8] C. M. Hoffmann and B. Yuan. On spatial constraint solving approaches. In *ADG '00: Proceedings of the 3rd International Workshop on Automated Deduction in Geometry*, volume 2061 of *Lecture Notes in Artificial Intelligence*, pages 1–15, Zürich, Switzerland, 2000. Springer.

[9] C. Jermann, G. Trombettoni, B. Neveu, and P. Mathis. Decomposition of geometric constraint systems: a survey. *International Journal of Computational Geometry and Applications*, 16(5,6):379–414, 2006.

[10] P. Mathis, P. Schreck, and R. Imbach. Decomposition of geometrical constraint systems with reparameterization. In *27th Symposium On Applied Computing*. ACM, 2012.

[11] P. Mathis and S. E. B. Thierry. A formalization of geometric constraint systems and their decomposition. *Formal Aspects of Computing*, 22(2):129–151, 2010.

[12] D. Michelucci and S. Foufou. Using Cayley-Menger determinants for geometric constraint solving. In *SMA '04: Proceedings of the 9th ACM symposium on Solid Modeling and Applications*, pages 285–290, Genoa, Italy, 2004. Eurographics Association.

[13] D. Michelucci and S. Foufou. Geometric constraint solving: The witness configuration method. *Computer-Aided Design*, 38(4):284–299, 2006.

[14] D. Michelucci and S. Foufou. Detecting all dependences in systems of geometric constraints using the witness method. In *ADG '06: Proceedings of the 6th international workshop on Automated Deduction in Geometry*, volume 4869 of *Lecture Notes in Artificial Intelligence*, pages 98–112, Pontavedra, Spain, 2007. Springer.

[15] D. Michelucci and S. Foufou. Interrogating witnesses for geometric constraint solving. In *SPM '09: Proceedings of the SIAM/ACM joint conference on Geometric and Physical Modeling*, pages 343–348, San Francisco, CA, U.S.A., 2009. ACM.

[16] D. Michelucci, P. Schreck, S. E. B. Thierry, C. Fünfzig, and J.-D. Génevaux. Using the witness method to detect rigid subsystems of geometric constraints in CAD. In *SPM '10: Proceedings of the ACM Conference on Solid and Physical Modeling*, pages 91–100, Haïfa, Israël, September 2010. ACM.

[17] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[18] J. Palsberg and C. B. Jay. The essence of the visitor pattern. In *Proceedings of the 22nd International Computer Software and Applications Conference*, COMPSAC '98, pages 9–15, Washington, DC, USA, 1998. IEEE Computer Society.

[19] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C, 2nd Edition*. Cambridge University Press, 1992.

[20] J. Richter-Gebert. *Realization Spaces of Polytopes*, volume 1643 of *Lecture Notes in Mathematics*. Springer, 1996.

[21] K. Stephenson. Circle packing: A mathematical tale. *Notices of the ACM*, 50(11):1376–1388, 2003.