Extensions of the witness method to characterize under-, over- and well-constrained geometric constraint systems

Simon Thierry, Pascal Schreck, Dominique Michelucci, Christoph Fünfzig, Jean-David Génevaux

Extensions of the witness method to characterize under-, over- and well-constrained geometric constraint systems

Simon E.B. Thierry[a,*], Pascal Schreck[a], Dominique Michelucci[b], Christoph Fünfzig[b], Jean-David Génevaux[a]

[a] *LSIIT, UMR CNRS 7005*
Université de Strasbourg
[b] *LE21, UMR CNRS 5158*
Université de Bourgogne

Abstract

This paper describes new ways to tackle several important problems encountered in geometric constraint solving, in the context of CAD, and which are linked to the handling of under- and over-constrained systems. It presents a powerful decomposition algorithm of such systems.

Our methods are based on the witness principle whose theoretical background is recalled in a first step. A method to generate a witness is then explained. We show that having a witness can be used to incrementally detect over-constrainedness and thus to compute a well-constrained boundary system. An algorithm is introduced to check if anchoring a given subset of the coordinates brings the number of solutions to a finite number.

An algorithm to efficiently identify all maximal well-constrained parts of a geometric constraint system is described. This allows us to design a powerful algorithm of decomposition, called $\mathcal{W}$-decomposition, which is able to identify all well-constrained subsystems: it manages to decompose systems which were not decomposable by classic combinatorial methods.

Key words: Geometric Constraints Solving, Witness configuration, Jacobian matrix, $\mathcal{W}$-decomposition, Under-constrainedness, Over-constrainedness, Well-constrainedness, Transformation groups

1. Introduction

Geometric constraint solving in Computer-Aided Design (CAD) aims at yielding a figure which meets some incidence and metric requirements (*e.g.* distances between points or angles between lines), usually specified in graphical

*Correspondence should be addressed to this author
Email addresses: simon.thierry@unistra.fr (Simon E.B. Thierry), schreck@unistra.fr (Pascal Schreck), dmichel@u-bourgogne.fr (Dominique Michelucci), c.fuenfzig@gmx.de (Christoph Fünfzig), jean-david.genevaux@etu.unistra.fr (Jean-David Génevaux)

form. Formally, a geometric constraint system (GCS) consists of a 3-tuple (C, X, A) with C a set of constraints (predicates) on a set X of unknowns (geometric elements) with respect to a set A of parameters (metric values). Solutions are returned as the coordinates of the geometric elements, *i.e.* a set of functions $f : X \to \mathbb{R}^i$ in i dimensions. For more formal definitions of geometric constraint systems, the reader may refer to [29].

Example : Geometric constraint system

Figure 1 shows a classical example of a geometric constraint system. We give a formal statement of the system and a sketch as the user would draw it. The right of the figure shows a possible solution. Many other solutions exist:

- if a symmetry is applied on a solution, it yields another solution;
- if a rotation and/or a translation is applied on a solution, it yields another solution;
- if a symmetry is applied for instance on point p_3, with axis p_2p_4, it yields another solution.

$\text{dist}(p_1, p_2, k_1)$, $\text{ang\_ppp}(p_6, p_1, p_2, \theta_1)$,
$\text{dist}(p_2, p_3, k_2)$, $\text{ang\_ppp}(p_2, p_3, p_4, \theta_2)$,
$\text{dist}(p_3, p_4, k_3)$, $\text{ang\_ppp}(p_4, p_5, p_6, \theta_3)$,
$\text{dist}(p_4, p_5, k_4)$, $\text{dist}(p_5, p_6, k_5)$,
$\text{dist}(p_1, p_6, k_6)$, with $k_1 = 7$, $k_2 = 5$,
$k_3 = 9$, $k_4 = 8$, $k_5 = 6$, $k_6 = 7$, $\theta_1 = 135$,
$\theta_2 = 120$, $\theta_3 = 115$

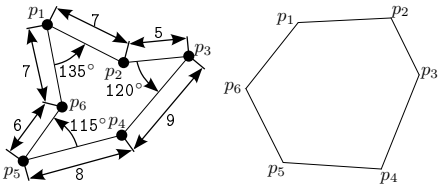

Figure 1: Formal statement (left) of a 2D technical sketch (middle) and a possible solution (right).

The literature describes a number of different approaches to solve geometric constraint systems:

- algebraic methods consist in translating the GCS into a set of equations and working on the equation system, thus forgetting the geometrical background. Algebraic methods can be classified in numerical methods (iterative computations converging to an approximate solution from initial values given by the user, such as the Newton-Raphson or the continuation method [27]) and symbolic methods (direct computations on the equations – those methods are seldom used because of their time complexity [1]),
- geometric methods use the geometric knowledge to solve the system: rule-based methods [2, 19] deduce construction plans by an explicit use of geometric rules, graph-based methods [7, 10, 27, 33, 34, 38] compile this knowledge into algorithms which consider only combinatorial and connectivity criteria,

- hybrid methods [5, 9, 20] alternate algebraic and geometric phases of computations to use the power of both approaches.

For more details on geometric constraint solvers, see [13]. A general trend, both to reduce complexity and to enhance resolution power, consists in decomposing the GCS into solvable subsystems and in assembling their solutions [5, 10, 14, 17, 27, 33, 34, 38, 42].

Example : Decomposition of geometric constraint systems

It is impossible to draw directly a solution of the 2D example of Figure 1 with only a ruler and a compass. But it is easy to separately solve each "triangle" ($p_1p_2p_6$, $p_2p_3p_4$ and $p_4p_5p_6$) and then assemble them. By using decomposition, the resolution power is thus greater.

Notice that, on the example of Figure 1, if one removes one of the triangles, say $p_2p_3p_4$, and then tries to solve the remaining system, one needs to add information from the solved subsystem, *i.e.* a distance constraint between p_2 and p_4, otherwise the remaining system becomes articulated.

For a detailed survey of decomposition methods, see [18]. The piece of information added when removing a subsystem is called a boundary [29]. Although several methods exist to find the relevant information in specific resolution frameworks [33], no general algorithm yet exists to compute the boundary without adding too much information.

Indeed, it is important for resolution methods, especially for graph-based methods, that the system does not have too few or too many constraints. Loosely speaking, a system is called

- under-constrained if it has an infinite number of solutions because there are not enough constraints to pin down every geometric element,
- over-constrained if it has no solution because of constraint contradictions[1],
- well-constrained if it has a finite positive number of solutions.

Invariance of rigid systems by direct isometries is generally taken into account by anchoring a point and a direction in 2D, a point and two directions in 3D. The point and the direction are called a *reference* for the direct isometries and constitute what we call an *anchor* of the system. Other transformation groups may be considered [37, 42]: we say that a system is G-well-constrained if it is well-constrained *modulo* G [29].

[1] Notice that the definitions of the levels of constrainedness are general and do not take into account the genericity hypothesis, further discussed in section 2. Thus, there exist systems which are said to be consistently over-constrained, when they are generically over-constrained but the values of the parameters are such that there are solutions.

Example : Transformation groups

The system of Figure 1 is rigid, or well-constrained *modulo* direct isometries. The system of Figure 2a is well-constrained *modulo* direct isometries, but contains a subsystem, shown at Figure 2b, which is well-constrained *modulo* similarities [37]: one can yield a finite number of solutions from which any solution can be generated by applying a scaling, a rotation and/or a translation.

The system of Figure 2c is under-constrained *modulo* any group acting globally on the system: if one anchors points p_1 and p_2, for instance, points p_3 and p_4 may still move without violating constraints.

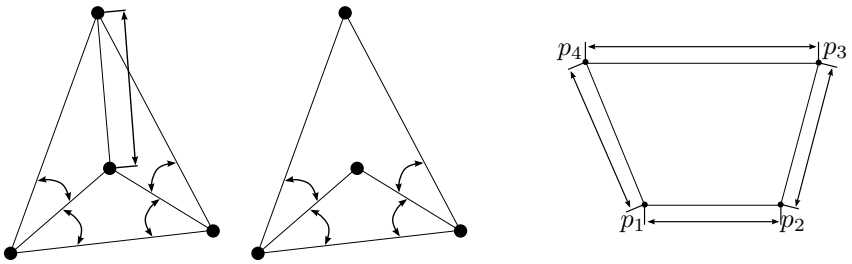

Figure 2: A rigid system (left) ; a subsystem well-constrained *modulo* similarities (middle) ; an articulated system (right)

A lot of work has been done about the detection of over-constrainedness [15, 17, 32] or under-constrainedness [21, 41, 46] and more generally about the characterization of rigidity [22, 24, 37, 44]. Yet, methods described in the literature may fail to consider the consequences of mathematical theorems that are not explicitly taken into account in the construction of the resolution algorithm. Since a theorem list cannot be exhaustive[2], it is impossible to develop a rule-based or graph-based algorithm which detects all geometric properties induced by mathematical theorems.

In this article, we extend the witness method [30] to address several problems cited above: how to determine the constrainedness level of a GCS without being tricked by mathematical theorems (see for instance Figure 11); how to build a well-constrained boundary system; how to check if a potential anchor does not make the system over-constrained; how to efficiently detect all maximal well-constrained subsystems of a given GCS; how to decompose a well-constrained system into the set of all its minimal well-constrained subsystems.

For conciseness reasons, in the rest of this paper, we consider 2D systems, unless explicitly mentioned otherwise. Yet, all algorithms can be extended to 3D systems with nearly no changes.

This article is organized as follows: Section 2 recalls the principles of the witness method and gives a way to generate a witness; Section 3 introduces an

[2] More precisely, the set of theorems is recursively enumerable, but not recursive in general.

incremental version of the Gauss-Jordan elimination method and demonstrates that it leads to a correct greedy algorithm to compute a well-constrained boundary system; Section 4 shows how to decide if a potential anchor is valid or not; Section 5 gives algorithms to efficiently identify the maximal well-constrained subsystems of an articulated system (also called flexible system); Section 6 deduces from these algorithms a method to further decompose a rigid system into well-constrained subsystems; Section 7 discusses the robustness issues of our algorithms; finally, Section 8 concludes and gives perspectives to this work.

2. The witness method

2.1. Principle

The notion of witness appears in different domains such as the study of polynomial systems through the principle of algebraic probability one [39], probabilistic proofs in geometry [6] or the Rigidity Theory [11].

The idea consists in studying generic properties of a continuous collection of objects through the study of a single one of these objects: a witness. Since the rigidity is an important part of our concern, we recall here the basics of the rigidity theory as described in [11].

2.1.1. Frameworks and rigidity

The question of rigidity is studied through the notion of frameworks. A *framework* is a triple (V, E, p) where (V, E) is a graph and $p : V \to \mathbb{R}^d$ a realization of the graph, which maps the vertices of V to points of dimension d. Thinking of graph edges as rigid bars and of vertices as articulation points, the main goal of combinatorial rigidity is to answer "Is (V, E, p) rigid?", *i.e.* are rigid body motions allowed only on the whole framework, with no local deformation.

An *infinitesimal flexion* is then a map $q : V \to \mathbb{R}^d$ such that $(p(i) - p(j)) \cdot (q(i) - q(j)) = 0$, for each $(i, j) \in E$. A framework is called *infinitesimally rigid*, if the only infinitesimal flexions arise from the direct isometries of $\mathbb{R}^d$, *i.e.* the translations and rotations. It is proven that infinitesimal rigidity is a stronger property than rigidity: a framework can be rigid but not infinitesimally rigid, famous examples are given in Figure 3 (see [11]). Counter-examples of rigid but not infinitesimally rigid frameworks arise when the framework is *singular*.

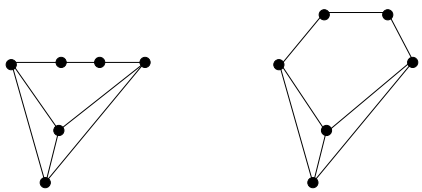

Figure 3: Non-infinitesimally rigid frameworks. The framework on the left is rigid.

A framework $F = (V, E, p)$ is said *generic* if there is a neighborhood of F where all frameworks with graph (V, E) are rigid if F is, and not rigid otherwise.

A generic rigid framework is said generically rigid. Two main results in rigidity theory are stated by the following proposition:

Proposition 1. *Consider a graph (V, E). If there is a realization p such that the framework (V, E, p) is generically rigid, then the frameworks (V, E, q) where q is another realization of the graph, is rigid for almost any q. On the other hand, if two frameworks (V, E, p) and (V, E, q) are generic, then they are both rigid or both not rigid.*

This proposition justifies the fact that in 2D, the Laman theorem [24] gives a combinatorial characterization of rigidity. Alas, such a characterization is an open problem in dimension 3 or higher.

From the geometric constraint point of view, a framework in rigidity theory corresponds to the realization of a geometric constraint system where all constraints are point-to-point distance constraints: such a system is generically well-constrained up to direct isometries if it is generically rigid. This was generalized by Michelucci *et al.* [30, 31] to metric constraints over points, lines, *etc.* (distances and angles) and to incidence constraints (colinearities in 2D and 3D, coplanarities in 3D).

2.1.2. Extension to CAD

In the previous section, we have recalled the basics of rigidity theory and the principles of witness interrogation in this context. In the rest of this article, we do not have the point of view of rigidity theory and of frameworks, but focus on geometric constraint system.

As stated above, geometric constraint systems in CAD naturally lead to constraint graphs, or more generally to hypergraphs. It is then tempting to extrapolate results of the rigidity theory, such as Laman's theorem, into the field of constraint solving. In our case, the constraints are put in a graphical form on the sketch (see Figure 1) which is a realization of the constraint graph the same way as in Rigidity Theory. Under some genericity assumptions, it is a perfect candidate to be a witness for the constrainedness properties.

Indeed when the designer draws a sketch, he/she has a solution X_w for an equation system $F(X, A_w) = 0$, with some parameter values A_w read on the sketch. Then the goal is a solution for the system $F(X, A_a) = 0$, where A_a are the values given for the dimensioning. This fact has been used within the continuation method with homotopy in CAD [3, 25] or to define a neighborhood relationship between figures [4]. In fact, our purpose is close to these problematics since we claim that the sketch is like the searched solution from the constrainedness point of view.

In the CAD domain, all the geometric constraints can be put under the form of polynomial equations, and F is a C^∞ class function. We can then consider a Taylor expansion of system $F(X, A_w) = 0$, and get:

$$F(X_w + \varepsilon\overrightarrow{V}, A_w) = F(X_w, A_w) + \varepsilon F'(X_w, A_w)\overrightarrow{V} + \mathcal{O}(\varepsilon^2)$$

where $\overrightarrow{V}$ can also be seen as the instant velocity of each object involved in the system and ε is a small time step. Then, $\varepsilon\overrightarrow{V}$ is an infinitesimal flexion, or motion, if it leads from a solution to another solution of the system. Or, in other words, the $\mathcal{O}(\varepsilon^2)$ term in the previous formula becomes in fact a $o(\varepsilon^2)$ term. Under these conditions, we must have

$$F'(X_w, A_w)\overrightarrow{V} = 0 \tag{1}$$

The space of the infinitesimal motions allowed by the constraints at the witness is then given by $\ker(F'(X_w, A_w))$. Note that

- the matrix $F'(X_w, A_w)$ is known as the Jacobian matrix of the function $F(X, A_w)$ taken at point X_w;
- when all constraints are point-to-point distances, the Jacobian matrix is the rigidity matrix considered in Rigidity Theory;
- for other constraints with parameters the genericity conditions are alike those in the combinatorial case: a parameter value A_w and a corresponding solution X_w are generic if the root is an implicit function of the parameters in some open neighborhood of (X_w, A_w); for instance, for a triangle specified with three length parameters, this condition forbids that one length is the sum of the others; more generally this condition implies that the matrix

  $$\begin{pmatrix} \partial F(X,A)/\partial X & \partial F(X,A)/\partial A \\ 0 & Id \end{pmatrix}$$

  has the same rank in an open neighborhood of (X_w, A_w). When all the equations are polynomials, because of the algebraic probability one principle, the generic parameter values are dense in the set of parameter values corresponding to a realization.

Example : Generic formulation of constraints

For point, line, plane incidences, we assume that the corresponding constraints are specified explicitly without parameters. This is to avoid expressing point-point incidences by a distance constraint $(p_{1,x} - p_{2,x})^2 + (p_{1,y} - p_{2,y})^2 = d^2$ with distance parameter $d = 0$. For a distance constraint $(p_{1,x} - p_{2,x})^2 + (p_{1,y} - p_{2,y})^2 = d^2$, the parameter $d = 0$ is not generic, as the constraint is singular at the solution point. For an angle constraint $\text{angle}(p_1, p_2, p_3) = \theta$, *i.e.* $\overrightarrow{p_2p_1} \cdot \overrightarrow{p_2p_3} = l_{p_1p_2} l_{p_3p_2} \cos\theta$, the parameter values $\theta = \pm\pi$, $\theta = \pm\pi/2$, and $\theta = 0$ are not generic. Similarly, point-line, line-plane incidences and line-line, plane-plane parallelism/orthogonality constraints are not expressed by angle constraints because it would introduce non-generic angles.

Typicality. A witness is *typical* if it is representative for the searched solution, *i.e.* it has the same combinatorial properties (coincidences, colinearities, coplanarities, *etc.*). [4] assume that the sketch is a witness, and that the sketch

and the witness are on the same continuation path. But even a random solution (X_w, A_w), $\{(X, A) : F(X, A) = 0\}$ with the specified combinatorial properties is typical with probability 1 for a set of witness solutions. Note that [18] built an (artificial) counter-example, *i.e.* a system with two classes of solutions (thus with two kinds of witnesses) which are different in combinatorial properties, and no continuous deformation exists to transform one into the other. In such cases, a witness is representative of only one class (its class) of solutions. This example is the one of Figure 4. Such artificial systems are ignored in this article.

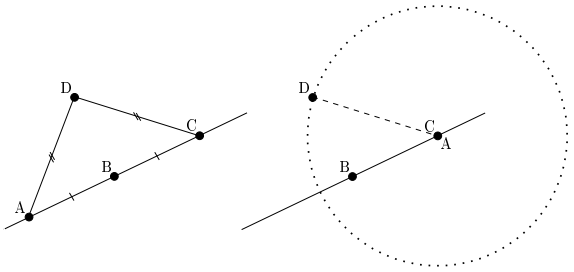

Figure 4: Ambiguous system: if points are required to be distinct, then it is rigid (left case); otherwise, when A and C have the same coordinates, point D can be anywhere on the dotted circle (example taken from [18, Figure 14]).

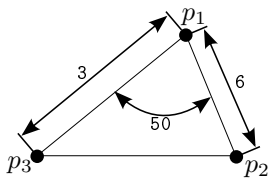

Figure 5: A 2D sketch of a rigid triangle.

We can then study the degrees of freedom of the system by studying the rank of the Jacobian matrix $F'(X_w, A_w)$ on a typical witness X_w, and in the case of under-constrainedness, the structure of the allowed infinitesimal motions can be deduced from the study of the kernel of $F'(X_w, A_w)$.

In the rest of this paper, we consider that rows of the Jacobian matrix represent constraints and columns represent values of the unknowns (coordinates of the geometric elements). We classically denote by m the number of rows and by n the number of columns of the matrix.

2.2. Practical example

Let us now detail a step-by-step construction of a witness and its use to characterize the degrees of freedom of the corresponding geometric constraint system. We consider the trivial geometric constraint system of Figure 5.

This GCS consists of three sets:

- unknowns: $X = \{p_1, p_2, p_3\}$;
- parameters: $A = \{k_1, k_2, \alpha\}$;
- constraints: $C = \{\text{dist}(p_1, p_2, k_1), \text{dist}(p_2, p_3, k_2), \text{ang\_ppp}(p_1, p_2, p_3, \alpha)\}$

The dimensioning given by the sketch is a function $\rho : A \to \mathbb{R}$, with $\rho(k_1) = 6$, $\rho(k_2) = 4$, $\rho(\alpha) = 50$. If one were to consider the bottom left corner of the sketch as the origin and give coordinates to the points on the sketch, they would not satisfy the constraints with this dimensioning.

Table 1: The Jacobian matrix for the system of Figure 1.

	$\dot{x_1}$	$\dot{y_1}$	$\dot{x_2}$	$\dot{y_2}$	$\dot{x_3}$	$\dot{y_3}$
r_1:dist(p_1, p_2, k_1)	x_1-x_2	y_1-y_2	x_2-x_1	y_2-y_1	0	0
r_2:dist(p_2, p_3, k_2)	0	0	x_2-x_3	y_2-y_3	x_3-x_2	y_3-y_2
r_3:ang_ppp(p_1, p_2, p_3, α)	$x_3 - x_2$	$y_3 - y_2$	$2x_2 - x_1 - x_3$	$2y_2 - y_1 - y_3$	$x_1 - x_2$	$y_1 - y_2$

Table 2: The Jacobian matrix of Table 1 at a witness.

	$\dot{x_1}$	$\dot{y_1}$	$\dot{x_2}$	$\dot{y_2}$	$\dot{x_3}$	$\dot{y_3}$
r_1	-1.1	2.58	1.1	-2.58	0	0
r_2	0	0	4.4	0	-4.4	0
r_3	-4.4	0	5.5	-2.58	-1.1	2.58

Let us now measure the distances on the sketch. We obtain a function ρ', with $\rho'(k_1) = 2.82$, $\rho'(k_2) = 4.35$, $\rho'(\alpha) = 70.9$. Obviously, the coordinates of the points on the sketch satisfy the constraints of the system if the dimensioning considered is ρ'. The genericity hypothesis tells us that the combinatorial properties of this alternate GCS are the same as the one with the dimensioning imposed by the user. For instance, if a subsystem of the GCS with ρ' is rigid, then it is also rigid with ρ.

Let us now compute the Jacobian matrix associated with the system. First, we need to translate the geometric constraints into equations. As explained above, a distance of k between two points p_i and p_j corresponds to the equation $(x_i - x_j)^2 + (y_i - y_j)^2 - k^2 = 0$. An angle of $\widehat{p_1 p_2 p_3}$ whose cosine is equal to a corresponds to the equation $(x_1 - x_2)(x_3 - x_2) + (y_1 - y_2)(y_3 - y_2) - a \times l_{p_1 p_2} \times l_{p_2 p_3} = 0$, with $l_{p_i p_j}$ indicating the distance between p_i and p_j.

By derivating those equations, we obtain the Jacobian matrix of the system, given at Table 1. Using the genericity hypothesis, we can replace the values of the unknowns by the coordinates taken on the sketch: $p_1 = (3.7, 2.98)$; $p_2 = (4.8, 0.4)$; $p_3 = (0.4, 0.4)$. This provides us with the Jacobian at the considered witness, shown on Table 2.

2.3. Generation of a witness

Most of the time, the sketch drawn by the user is a witness. Also, CAD parts are rarely designed from scratch: usually, previous similar parts are re-used and modified; the parameters values are changed and tuned for a new design, but the constraint system is left unchanged. Thus, solutions to the previous CAD parts give a witness.

Sometimes, however, no witness is available. For instance, when there are many incidence or tangency constraints, the sketch may not fulfill them. It may also happen that no previous sketch is available, for instance when designing a part or a mechanism for the first time. In those cases, a witness has to be computed. A witness is a root of the system $F(X, A) = 0$, where both X and A are unknowns. See Section 7.3 for more details about systems containing incidence constraints.

For problems occurring in CAD-CAM, these systems are usually strongly under-constrained: the solution is a manifold, *e.g.* a curve, a surface, *etc.* Thus,

several methods are possible in order to generate a witness. For instance, the next section presents the most general method: it uses a complete solver, *i.e.* a solver which finds all solutions (in real space $\mathbb{R}^n$). Even in the case of strongly under-constrained systems, such a solver can easily be tuned to stop at the first found root, and to explore the search space in some random order, so that the first found root is a typical witness with probability 1. Notice that this tuned solver is still complete in the sense that if it finds no root, then it is a proof that there is none.

We mention now some incomplete but simple and fast solvers which can be used to generate a witness by exploiting the characteristics of a witness: under-constrainedness and genericity.

First of all, when one faces a sketch which does not completely fulfill all incidence constraints, it is possible to use said sketch to initiate a Newton-Raphson iteration[3], an homotopy [25], an optimization method like Levenberg-Marquardt, or Nelder-Mead simplex [35]. If no sketch is available, or if the previous method fails, it is possible to start the iteration with random values for X and A. Meta-heuristics [26] like genetic programming, particle swarm, *etc.* can also be used with likely success, due to the strong under-constrainedness of the system to be solved. With all those methods, approximation problems arise: this issue will be further discussed in section 7.

2.3.1. Using a complete solver to generate a witness

To compute a witness (X_w, A_w), we solve the under-determined system $\{(X, A) : F(X, A) = 0\}$ where both X and A are unknowns with a complete solver, the subdivision solver presented in [8].

The nonlinear monomials x_i^2 and $x_i x_j$ for $i < j$ are replaced by additional variables $x_{i,i}$ and $x_{i,j}$, which are enclosed in a polytope $B_D(x_i, x_{i,i}, x_{i,j,i<j}) \geq 0$ with halfspaces given by the non-negativity of relevant Bernstein polynomials (*Bernstein polytope*). The quadratic constraint system becomes a polytope $S(x_i, x_{i,i},\ x_{i,j,i<j}) \geq 0$ after rewriting into the additional variables $x_{i,i}$ and $x_{i,j}$. The subscript D of $B_D(x_i, x_{i,i},\ x_{i,j,i<j}) \geq 0$ indicates that this polytope depends on the domain D. In this way, bounds for the solution domain of quadratic polynomials can be expressed as two linear programs

$$\begin{array}{l} \min x_i \text{ and } \max x_i \\ S(x_i, x_{i,i}, x_{i,j,i<j}) \geq 0 \\ B_D(x_i, x_{i,i}, x_{i,j,i<j}) \geq 0 \end{array}$$

Domain bounds are computed by linear programming in order to reduce the current solution domain D. If the feasible set is empty, which is detected by linear programming, then the current domain box contains no solution. Otherwise, we can perform a sequence of reductions and bisections of domain boxes

[3] The techniques, *e.g.* singular value decomposition [35], to account for non-square Jacobian matrices and under-constrained systems are well known.

until the domain box $D = [\underline{x_1}, \overline{x_1}] \times \ldots \times [\underline{x_n}, \overline{x_n}]$ is δ-small: $(\overline{x_i} - \underline{x_i}) < \delta$ for all i. These δ-small boxes cover the solution set piecewise.

The subdivision solver requires a domain box to start the search. The intervals for generic parameter values of constraints are easy to find: angle parameters $\cos\theta$ ($\cos\theta$ instead of θ to avoid trigonometric functions in the solver) are in $[-1+\epsilon, -\epsilon]$ or $[\epsilon, 1-\epsilon]$ with a small, arbitrary ϵ; intervals for distance parameters d can be obtained from magnitude bounds of the point coordinates. Finding a bound on the magnitude of any root [45], would be necessary to prove that the system has no solution. For the problems here, a bound on the point coordinates is known beforehand.

In order to enumerate all solutions of a system, we used mid-bisection of the largest interval in [8], which minimizes the height of the exploration tree while cycling through dimensions. For the case of determining a single solution as fast as possible, the choice of the smallest interval (greater or equal δ) is beneficial as setting variables to values allowing solutions improves the effectiveness of the domain reduction step.

We select the next domain box (of smallest minimum side length greater than δ) for reduction and bisection at random. In this way, we find a solution box containing a random solution, and we take the box center projected onto the solution set as a witness.

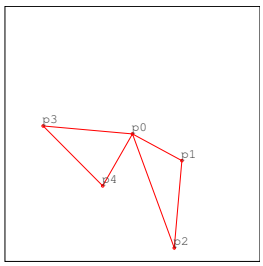

Figure 6: "The butterfly": 2D system with 5 points and 6 distance parameters $d(p_0, p_1)$, $d(p_1, p_2)$, $d(p_2, p_0)$, $d(p_0, p_3)$, $d(p_3, p_4)$, $d(p_4, p_0)$.

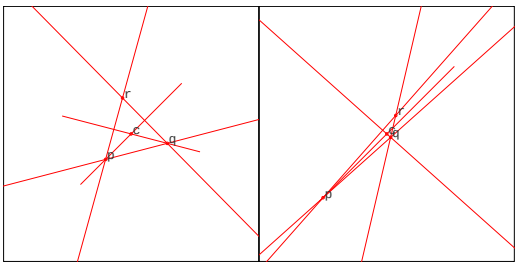

Figure 7: 2D system of 4 points and 5 lines with 10 point-line incidences, 4 angle parameter angle(qp, cp), angle(cp, rp), angle(rq, cq), angle(cq, pq) and 1 distance parameter $d(r, c)$. Symmetric solution (left) and random, typical witness solution (right).

As examples, we show two systems of different difficulty. In Figure 6, two triangles with a common point p_0 are specified by six side lengths. In the random solution, the side lengths are all different. In Figure 7, four points and five lines with 10 point-line incidences are specified by four angle parameters and a distance parameter. The left part shows a solution with symmetric and nice shaped triangles, obtained by additional minimum distance constraints between the triangle points. In the right part, a typical witness solution is shown, which was found at random. It is used for further analysis.

As a future work, we intend to consider using constraint propagation [16] to speed up the solver. Also, we intend to try other heuristics for the choice of the

unknown to bisect (for instance the smear function in [23]).

2.3.2. Discussion

One may question the usefulness of the witness method, when the complete solver has to be called to compute a witness. In this case, it seems that the complete solver is called two times instead of just one: why not simply call the complete solver on the target system (*i.e.* the system to be solved: $F(X, A_w) = 0$) ? The answers are three: first of all, the complete solver is most of the time much faster when it is used to compute a witness (assuming that it stops at the first found root), since one only needs to satisfy the incidence and tangency constraints; second, the witness will be used to analyze and decompose the GCS, which will usually speed up the second step: solving the target system; third, when the complete solver finds no root when computing a witness, it proves the absence of roots for every parameter values, *i.e.* the problem does not come from the parameters values A_w, but it has a more fundamental cause. Since the problem of debugging such systems of constraints occurs very rarely and is not the topic of this article, we just mention the principle of a solution: add incrementally each constraint and search a witness; it allows to find the first constraint which is the cause of the contradiction.

3. Incremental detection of redundancy

We already showed in Section 1 that the detection of over-constrainedness is a complicated yet essential problem in the field of geometric constraint solving.

In this section, we exhibit a greedy algorithm, based on the witness method, which allows an efficient and robust detection of redundancy in geometric constraint systems. We prove that this algorithm produces a minimal well-constrained subsystem with the same solutions as the initial one.

We also show the usefulness of this extension of the witness method to enhance robustness of decomposition methods by an accurate computation of the boundary system.

3.1. A greedy algorithm to detect redundancy in a GCS

It was already shown in [30] that it is possible to interrogate a witness in order to detect whether a set of constraints is dependent or not. Indeed, it is possible to compute the rank of the Jacobian matrix at the witness and to compare it with the sum of the degrees of restriction of the constraints. However, finding a maximal independent subset of a dependent set is not a trivial problem. Working on the witness, the naive idea would be to try and remove constraints one by one and, at each step, compute the rank again to determine if the constraint is redundant with the remaining set. If the rank computed for $\mathcal{S}-c$ equals the rank computed for $\mathcal{S}$, then constraint c is redundant and can be removed. Performed that way, the removal of redundant constraints is expensive. Yet, considering an incremental construction of the geometric constraint system allows to identify

the set of redundant constraints with no additional costs in comparison to the basic detection of redundancy.

Indeed, consider a geometric constraint system $\mathcal{S}$ with no redundancy between the constraints. Applying the Gauss-Jordan elimination method on the Jacobian matrix at the witness leads to a matrix $J' = (I \quad P)$, I being a $m \times m$ diagonal matrix and P being a $m \times f$ matrix, with $f = n - m$ the number of actual degrees of freedom of the system. This method has a known complexity of $\mathcal{O}(\min(n, m)nm)$. Let us now consider a system $\mathcal{S}'$ with $\mathcal{S} \subset \mathcal{S}'$. In order to know if $\mathcal{S}'$ is generically over-constrained, one only needs to incrementally add the geometric elements and the constraints (bearing in mind that a constraint can only be inserted when the geometric elements it concerns are all in the system) of $\mathcal{S}' - \mathcal{S}$ to $\mathcal{S}$ and applying Gauss-Jordan again. Since the leftmost part of the matrix J' is diagonal, the number of operations is at most $2 \times \min(m, n) \times f$: for each row of I, each non-zero element of P must be multiplied and added to the new row. The number of operations is in fact smaller, since the number of zero elements in the new row is high.

Proceeding incrementally does not raise the number of operations: it only changes the order of the operations. Indeed, the classical Gauss-Jordan elimination method consists of column-by-column operations: for each column c, divide row c by $J_{c,c}$, then subtract $J_{r,c}$ times the new row from row r for every r, so that column c is a null vector except for the cth value. With the incremental calculus of the reduced row echelon form, one proceeds row by row: for each row r, subtract $J_{r,c}$ times row c for each $c < r$, then divide row r by $J_{r,r}$ so that the $r - 1$ first elements of row r are zero and the rth element is 1. Thus, the overall time complexity of the incremental computation of the reduced row echelon form of J is also of $\mathcal{O}(\min(n, m)nm)$. The pseudo-code for this incremental construction of a maximal independent subset of the constraints is given at Algorithm 1.

Note that for the test of line 8, we do not here make explicit the case where a constraint c corresponds to several rows $r_1 \dots r_k$, for instance a distance between two lines. In such a case, it is necessary to be able to rewrite the constraint system in order to remove one row only (say r_1) and replace the constraint c with (an)other constraint(s) which correspond(s) to rows $r_2 \dots r_k$.

The incremental version of the Gauss-Jordan elimination has the same complexity as the one-step version, but has a major advantage in our case: at each step, when a constraint is inserted, one may compare the new rank with the previous one and thus detect a redundant constraint. With exactly the same number of operations as in the case of the classical Gauss-Jordan elimination, one obtains the reduced row echelon form of the Jacobian matrix together with the list of redundant constraints.

Let us now show that the order in which the constraints are considered in the incremental construction of a maximal independent subset does not change the solution set. Said otherwise, if there are several maximal non-over-constrained subsystems, they are equivalent. Following [29], we note $\mathcal{F}_\varphi(\mathcal{S})$ the solution set of a GCS for a valuation φ of the parameters, and omit φ when there is no ambiguity.

Algorithm 1: Greedy algorithm to compute a maximal non-over-constrained subsystem

Input:
$\mathcal{S}' = (C, X, A)$: a geometric constraint system
$\mathcal{W}$: a typical witness of $\mathcal{S}'$
Result: $\mathcal{S} \subseteq \mathcal{S}'$: maximal non-over-constrained subsystem of $\mathcal{S}'$

$R \leftarrow \emptyset$ `// set of redundant rows`
$J \leftarrow$ Jacobian matrix of $\mathcal{S}'$ at $\mathcal{W}$ (of size $m \times n$)
$J' \leftarrow$ empty matrix with no rows and n columns
foreach *row* r *of* J **do**
5 add r to J' (we call r' the new row of J')
   foreach *row* i *of* J' *except* r' **do**
7 $r' \leftarrow J'_{r',i} \times i$
8 **if** r' *is a null row* **then**
   `// the constraint corresponding to row` r `is redundant`
   Remove row r' from J'
   $R \leftarrow R \cup \{r\}$
   else
   $r \leftarrow \frac{r}{J'_{r,r}}$
$A' \subseteq A \leftarrow$ set of parameters appearing only in constraints of R
return $\mathcal{S} = (C/R, X, A/A')$

Proposition 2. *Let $\mathcal{S} = (C, X, A)$ be a generically over-constrained GCS. We consider a valuation φ of the parameters which make $\mathcal{S}$ consistently over-constrained,* i.e. *such that $\mathcal{F}_\varphi(\mathcal{S}) \neq \emptyset$. Let $\mathcal{S}''$ be a basis of $\mathcal{S}$,* i.e. *a maximal non-over-constrained system such that $\mathcal{F}_\varphi(\mathcal{S}) = \mathcal{F}_\varphi(\mathcal{S}'')$. Let $\mathcal{S}'$ be the system obtained by applying Algorithm 1 on $\mathcal{S}$. Then, $\mathcal{F}(\mathcal{S}') = \mathcal{F}(\mathcal{S}'')$.*

Proof We argue by mutual inclusion.
Step 1. $\mathcal{F}(\mathcal{S}'') \subseteq \mathcal{F}(\mathcal{S}')$.

$\mathcal{S}'$ is obtained by discarding some constraints of $\mathcal{S}$, without adding any new constraint: $\mathcal{S}' \subseteq \mathcal{S}$. Thus, a solution of $\mathcal{S}$ satisfies all constraints of $\mathcal{S}'$: $\mathcal{F}(\mathcal{S}) \subseteq \mathcal{F}(\mathcal{S}')$.

Since we consider only parameters valuations such that $\mathcal{F}(\mathcal{S}) = \mathcal{F}(\mathcal{S}'')$, we have $\mathcal{F}(\mathcal{S}'') \subseteq \mathcal{F}(\mathcal{S}')$.
Step 2. $\mathcal{F}(\mathcal{S}') \subseteq \mathcal{F}(\mathcal{S}'')$.

We have to show that the solutions of $\mathcal{S}'$ satisfy all the constraints of $\mathcal{S}''$, *i.e.* that a solution of $\mathcal{S}'$ is also a solution of $\mathcal{S}''$. We argue by contradiction.

Let us consider a row r of the Jacobian matrix $J_{\mathcal{S}''}$ of $\mathcal{S}''$ which is not redundant with the Jacobian matrix $J_{\mathcal{S}'}$ of $\mathcal{S}'$. If we try to add r to $J_{\mathcal{S}'}$ as we do at lines 5–7 of Algorithm 1, it will thus not be null at line 8.

Since $J_{\mathcal{S}'}$ was obtained by applying Algorithm 1 on the Jacobian matrix $J_{\mathcal{S}}$ of $\mathcal{S}$, row r is not redundant with $J_{\mathcal{S}}$ either: it thus corresponds to a constraint

Table 3: The Jacobian matrix for the system of Figure 8.

	$\dot{x_1}$	$\dot{y_1}$	$\dot{x_2}$	$\dot{y_2}$	$\dot{x_3}$	$\dot{y_3}$	$\dot{x_4}$	$\dot{y_4}$
r_1: dist(p_1, p_2)	x_1-x_2	y_1-y_2	x_2-x_1	y_2-y_1	0	0	0	0
r_2: dist(p_1, p_3)	x_1-x_3	y_1-y_3	0	0	x_3-x_1	y_3-y_1	0	0
r_3: dist(p_2, p_4)	0	0	x_2-x_4	y_2-y_4	0	0	x_4-x_2	y_4-y_2
r_4: dist(p_3, p_4)	0	0	0	0	x_3-x_4	y_3-y_4	x_4-x_3	y_4-y_3
r_5: dist(p_2, p_3)	0	0	x_2-x_3	y_2-y_3	x_3-x_2	y_3-y_2	0	0
r_6: dist(p_1, p_4)	x_1-x_4	y_1-y_4	0	0	0	0	x_4-x_1	y_4-y_1

which is not in $\mathcal{S}$ and cannot be deduced by the constraints of $\mathcal{S}$. We then have $\mathcal{F}(\mathcal{S}) \neq \mathcal{F}(\mathcal{S}'')$. By definition of $\mathcal{S}''$, this is not possible. There is a contradiction, and we thus have $\mathcal{F}(\mathcal{S}') \subseteq \mathcal{F}(\mathcal{S}'')$.
Conclusion. We have $\mathcal{F}(\mathcal{S}'') \subseteq \mathcal{F}(\mathcal{S}')$ and $\mathcal{F}(\mathcal{S}') \subseteq \mathcal{F}(\mathcal{S}'')$. Thus, $\mathcal{F}(\mathcal{S}') = \mathcal{F}(\mathcal{S}'')$.

□

Notice that the constraints which are identified as redundant may be kept and be used later in order to find, among the different solutions satisfying the constraints, which correspond best to the user's intent: redundancy can be necessary to ensure solution unicity [12].

3.2. Examples

We give here a few examples of the application of Algorithm 1.

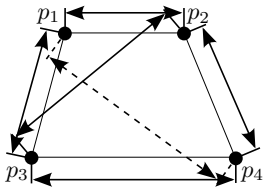

Figure 8: "The kite": over-constrained 2D system with 4 points and 6 distances. Without the dotted constraint, the system is rigid.

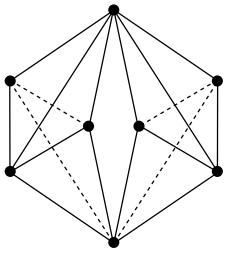

Figure 9: "The double-banana": famous counter-example to the extension of Laman's characterization of rigidity in 3D. Each segment represents a distance constraint.

Let us consider the 2D example of Figure 8. The Jacobian matrix of this system is shown in Table 3. Consider the following witness: $p_1 = (2, 7)$, $p_2 = (5, 6)$, $p_3 = (1, 1)$ and $p_4 = (6, 3)$. The Jacobian at this witness is shown in Table 4, with a partial Gauss-Jordan elimination, since the sixth row has not been modified. That is, Table 4 shows the matrix obtained by performing the incremental version of the Gauss-Jordan elimination, after inserting the sixth constraint but before performing Gauss pivoting on it, *i.e.* at the end of line 5 of the algorithm. It is easy to see that the sixth row is redundant, since it can

Table 4: The Jacobian matrix of Table 3 at a witness. The Gauss-Jordan elimination method was used on the first five rows. The sixth row is redundant ($r_6 = r'_2 - r'_1$)

	$\dot{x_1}$	$\dot{y_1}$	$\dot{x_2}$	$\dot{y_2}$	$\dot{x_3}$	$\dot{y_3}$	$\dot{x_4}$	$\dot{y_4}$
r'_1	1	0	0	0	0	$-\frac{4}{5}$	-1	$\frac{4}{5}$
r'_2	0	1	0	0	0	$-\frac{4}{5}$	0	$-\frac{1}{5}$
r'_3	0	0	1	0	0	$-\frac{3}{5}$	-1	$\frac{3}{5}$
r'_4	0	0	0	1	0	$-\frac{1}{5}$	0	$-\frac{4}{5}$
r'_5	0	0	0	0	1	$\frac{2}{5}$	-1	$-\frac{2}{5}$
r_6	-1	1	0	0	0	0	1	-1

be obtained by subtracting the first row from the second one. Thus, we detected the over-constrainedness.

For a more complex and famous example, consider the 3D system of the double-banana (see Figure 9). Since the associated Jacobian matrix is a 18×24 matrix, we do not represent it here, but applying Algorithm 1 on the first 17 rows and adding the last constraint of the double-banana leads to a zero-filled row in the Jacobian matrix at the witness. If one considers a variant of the double-banana with higher connectivity [28], our method still succeeds to efficiently detect over-constrainedness: the degree of connectivity of the constraint graph has no influence on the witness method. Likewise, the 3D examples by Ortuzar (see Figure 10) are correctly detected as over-constrained.

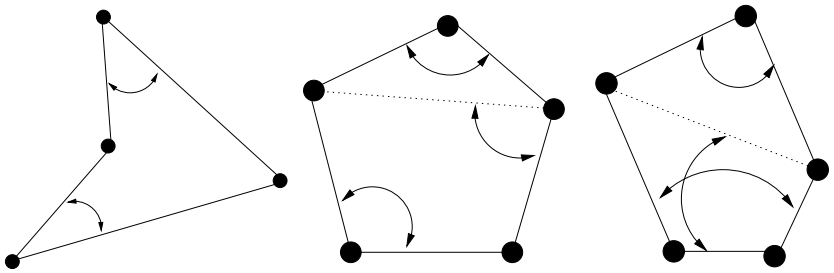

Figure 10: 3D examples (courtesy of Auxkin Ortuzar, Dassault Systèmes) which confuse graph-based methods but are detected over-constrained by our method. No three points are coplanar, plain segments represent distance constraints and arcs represent angle constraints.

Moreover, the witness method correctly handles redundancy in under-constrained cases, where graph-based methods are helpless because they do not consider geometric theorems. For instance, consider the 2D example of Figure 11. It is unlikely that a graph-based method can ever detect the fact that point y is fixed, no matter what coordinates are given to point p and line l. Hence, a graph-based method would see this system as a system with 8 remaining degrees of freedom (5 for the three aligned points a, b and x, 1 for line l traversing x and 2 for point p) and would consider that adding a constraint distance between points a and y removes a degree of freedom. The witness method, however, detects that this new distance constraint is redundant and that the unknown y is

determined by the system though l and p can be chosen at random.

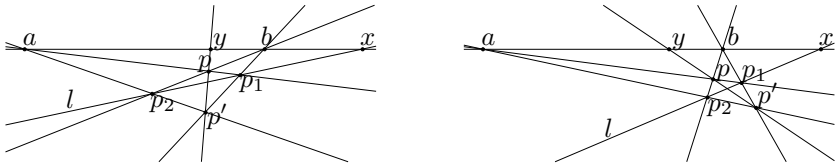

Figure 11: In 2D, given three aligned points a, b and x and for any point p and line l traversing x, y is unchanged: $p_1 = (ap) \cap l$, $p_2 = (bp) \cap l$, $p' = (ap_2) \cap (bp_1)$, $y = (ab) \cap (pp')$.

3.3. Computation of a well-constrained boundary system

The witness method and, more specifically, Algorithm 1 can be used to address an important problem in geometric constraint systems decomposition methods: the computation of a non-over-constrained boundary system.

We saw in the example of page 3 about decomposition, that it is important to be able to add information about a subsystem into the rest of the system, when using decomposition methods. The piece of information added is called *boundary*.

Recall [29] that a boundary system of a system $\mathcal{S}_1 \subseteq \mathcal{S}$ with respect to the system $\mathcal{S}_2 = \mathcal{S} - \mathcal{S}_1$ is a system B such that solutions in $\mathcal{F}(\mathcal{S}_2 + B)$ are all subfigures of a figure in $\mathcal{S}$. Said differently, it is a system which can replace $\mathcal{S}_1$ without modifying the solutions when one looks only at the coordinates of the geometric elements of $\mathcal{S}_2$. In decomposition methods [18], the computation of a boundary system is essential since without it, the recombination of subfigures can lead to figures which are not solutions of the system. Although not called "boundary system", this notion is present in all decomposition methods (*e.g.* it is explicit in the FRONTIER solver [33] and corresponds to the virtual bond in Owen's method [34]).

Intuitively, the boundary system of $\mathcal{S}_1$ with regard to $\mathcal{S}_2$ consists of the system (C, X, A) with X the set of geometric elements shared by $\mathcal{S}_1$ and $\mathcal{S}_2$, and C (and A) the set of all geometric information (and corresponding metric values) which can be computed about elements of X in $\mathcal{S}_1$. However, this approach can lead to generically over-constrained boundary systems, even though the associated parameters make them consistently over-constrained. If the considered solving method is sensitive to generic over-constrainedness, as are all combinatorial solvers, this approach cannot be considered. The example of Figure 12 shows a basic example where a naive boundary computation leads to over-constrainedness: here, $\mathcal{S}_1$ is the system consisting in the $p_1 \dots p_4$ points and the distance constraints represented by thick lines; the dotted lines represent constraints (of any type) concerning one of the p_i points and one other geometric element, not in $\mathcal{S}_1$. $\mathcal{S}_1$ is a rigid system. Hence, computing all the information about the elements of $\mathcal{S}_1$ means computing, among other constraints, all pairwise distances between the points $p_1 \dots p_4$: this leads to computing the system of Figure 8 as the boundary of $\mathcal{S}_1$ with regard to the rest of the system, *i.e.* an over-constrained system.

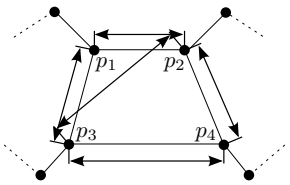

Figure 12: The boundary of the rigid subsystem with $X = \{p_1, p_2, p_3, p_4\}$ is generically over-constrained: it contains the system of Figure 8

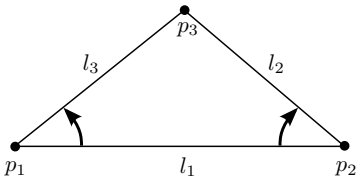

Figure 13: Triangle well-constrained *modulo* similarities with six incidence constraints and two line-line angle constraints

Using our greedy algorithm, computing a basis of the boundary system is easy: one only needs to start with an empty boundary system and add the different computable constraints one by one, discarding those which are redundant with the ones already considered. In order to get all computable constraints, one needs to know the transformation group G such that the system is well-constrained *modulo* G [37]: then, one only needs to compute all possible G-invariant constraints possible to express. For instance, if distances are G-invariant (*i.e.* G is the rotations, the translations or the direct isometries), one computes all point-point distances between two points of the system. The exact process is given at Algorithm 2. Proposition 2 proves that the order in which the constraints are considered does not matter.

Algorithm 2: Computation of a maximal non-over-constrained boundary system

Input:
$\mathcal{S} = (C, X, A)$: a geometric constraint system
$\mathcal{S}_1 = (C_1, X_1, A_1)$: a subsystem of $\mathcal{S}$
G: well-constrainedness group of $\mathcal{S}_1$
Result: B: maximal non-over-constrained boundary system of $\mathcal{S}_1$

$X_b \leftarrow \{x \in X_1 | x$ is concerned by a constraint $c, c \in C/C_1\}$
$C_b \leftarrow$ set of all possible G-invariant constraints concerning elements of X_b
$A_b \leftarrow$ set of parameters appearing in C_b `// Values computed from` $\mathcal{S}_1$
$B \leftarrow$ result of Algorithm 1 on (C_b, X_b, A_b)
return B

An additional problem is the fact that the geometrical universe considered by the solver may not allow to express the different constraints which can be computed in $\mathcal{S}_1$: for instance, it may be possible to compute, in $\mathcal{S}_1$, that a point p is incident to a line l, but if the geometrical universe does not include point-line incidence constraints, this constraint will not be considered. In such a case, it is not possible to ensure that the computed system is indeed a boundary system. This issue is discussed in [29].

4. Decision of anchor validity

4.1. Over-constraining anchors

Given a system $\mathcal{S}$ and a witness of $\mathcal{S}$, we can show using the witness method that it is not generically over-constrained or, if it is, compute a basis of $\mathcal{S}$, that is to say a maximal non-over-constrained system which has the same solution set as $\mathcal{S}$ when we consider valuation parameters which make $\mathcal{S}$ consistently over-constrained. Let us therefore consider systems which are not generically over-constrained.

$\mathcal{S}$ may be still under-constrained. Following the multi-group approach [37], the under-constrainedness may be the result of the well-constrainedness of the system *modulo* a transformation group: for instance, a rigid system is under-constrained, since it may be translated or rotated without violating any constraint. In [30], we show how to use the witness method to detect that a system is invariant under the action of translations and rotations, by recognizing these groups in the kernel of the Jacobian matrix (see in particular Table 1 of the article). One can, likewise, recognize scalings by simulating their action on two points. Hence, one may compute the dimension of the kernel of the Jacobian and, if it is higher than the number of invariance groups identified within it, conclude that the system is articulated: it is under-constrained even *modulo* global groups.

When dealing with an articulated GCS, one may want to determine which geometric elements should be anchored in order to get a finite number of solutions. This research of an anchor is a kind of parameterization of a geometric constraint system, since it consists in giving a list of coordinates which should be given as parameters and not considered as unknowns if one is to have a finite number of solutions. It generalizes the notion of G-reference [29]: an anchor for a G-well-constrained GCS is a G-reference.

We do not here discuss an algorithm to find a parameterization of an articulated GCS, but we provide a procedure to decide if a set of coordinates is a valid anchor. Indeed, a set of coordinates which has the same size as the dimension of the kernel is not necessarily an anchor for the system, and may be an *over-constraining anchor* if considering them as parameters leads to the absence of solutions.

For instance, consider a simple rigid triangle. It has three degrees of freedom *modulo* direct isometries, thus an anchor must contain three coordinates. A classical anchor would be both coordinates of one point, and one coordinate of another, but other anchors are valid as well: abscissas of two points and the ordinate of another. But the set of all abscissas is not a valid anchor, because in any rigid system, with two abscissas anchored, the abscissas of all points are fixed. Likewise, consider the system of Figure 13, which is well-constrained *modulo* similarities. Since any line-line angle can be computed in this system, no anchor containing the directions of two lines is valid (*e.g.* both coordinates of p_1, directions of l_1 and l_2).

4.2. A decision algorithm for over-constraining anchors

Over-constraining anchors are due to a dependency of the coordinates in the GCS, that is to say that given a subset of an over-constraining anchor and the constraints, one can compute the values of another subset of the anchor. We give here an algorithm to detect such dependences and hence decide whether a potential anchor is over-constraining or not.

Obviously, the size of an anchor must be exactly the dimension of the kernel of the Jacobian matrix of the system, *i.e.* the number of actual degrees of freedom. We thus do not discuss the cases of over-sized anchors (which are in any case over-constraining) or under-sized anchors (which can still be over-constraining and, when they are not, are not actual anchors either, since they do not lead to a finite number of solutions).

When considering a valid anchor, the $J\overrightarrow{V} = 0$ equation[4], once J put in reduced echelon form, is the following (for convenience we use here the notation v_i instead of $\dot{x}_i$):

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & \alpha_{1,1} & \cdots & \alpha_{n-m,1} \\ 0 & 1 & 0 & \cdots & 0 & \alpha_{1,2} & \cdots & \alpha_{n-m,2} \\ 0 & 0 & 1 & \cdots & 0 & \alpha_{1,3} & \cdots & \alpha_{n-m,3} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & \alpha_{1,m} & \cdots & \alpha_{n-m,m} \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_m \\ v_{a_{n-m}} \\ \vdots \\ v_{a_1} \end{pmatrix} = 0$$

In this notation, the $n-m$ last elements of vector $\overrightarrow{V}$ are denoted $v_{a_{n-m}}$ to v_{a_1}. There are as many v_{a_k} elements as the dimension of the kernel of J.

Let us now imagine that after the $i-1$th step of the Gauss-Jordan elimination (the upper left $(i-1)\times(i-1)$ matrix is diagonal), it is impossible to find a non-zero pivot. It means that the ith column is a linear combination of the first $i-1$ columns. Permuting this column with one of the first $i-1$ columns will not change the problem: the new ith column will (by definition of linear dependency) be a linear combination of the new leftmost $i-1$ columns. Likewise, permuting the ith column with a column of index between $i+1$ and m will only postpone the problem: if it is a linear combination of the first $i-1$ columns, it is also a linear combination of the first $i-1+k$ columns if $k \geq 0$.

It will thus not be possible to diagonalize the left part of the Jacobian matrix unless the ith column is permuted with one of the $n-m$ rightmost columns. This condition is necessary but not sufficient, since some of the last $n-m$ columns can themselves be linear combination of the first $i-1$ columns.

If one does not manage to diagonalize the $m \times m$ left matrix of the Jacobian, it means that the parameterization consisting in anchoring the coordinates cor-

[4] *Cf.* equation 1, p. 7.

responding to the $n-m$ rightmost columns is not valid: it means that it is not possible to express the variations of the n first coordinates as a function of those considered as parameters. Indeed, once the product $J\overrightarrow{V}$ performed, the ith row of the equation can be written

$$v_i + \alpha_{1,i} \times v_{a_{n-m}} + \cdots + \alpha_{n-m,i} \times v_{a_1} = 0$$

and thus

$$v_i = -\alpha_{1,i} \times v_{a_{n-m}} - \cdots - \alpha_{n-m,i} \times v_{a_1} \tag{2}$$

Said differently, we can express v_i (the variations of the coordinate x_i) as a function of $v_{a_{n-m}} \dots v_{a_1}$ (the variations of the coordinates $x_{a_{n-m}} \dots x_{a_1}$). This is the exact definition of an anchor: if the x_{a_k} coordinates are fixed, the other elements are also fixed. Thus, if it is not possible to express the first m columns as function of the $n-m$ last columns, then the corresponding parameterization is an over-constraining anchor.

Algorithm 3: Algorithm to decide if a given subset of the unknowns is a valid anchor

Input:
$\mathcal{S} = (C, X, A)$: a geometric constraint system
$\mathcal{W}$: a typical witness of $\mathcal{S}$
$\mathcal{A}$: a subset of the coordinates of the elements of X
Result: Boolean indicating if $\mathcal{A}$ is a valid anchor for $\mathcal{S}$

```
J ← Jacobian matrix of S at W (of size m × n)
k ← n
for i from 1 to m − |A| do
    while column i corresponds to a coordinate in A do
        Permute column i of J with column k
        k ← k − 1
    if there is a non-null pivot in column i then
        Perform Gauss-Jordan elimination on column i
    else
        return false
return true
```

A simple method to decide if a given subset of the coordinates forms a valid anchor is thus to do the necessary permutations in order to have the corresponding columns at the right of the matrix, and then to perform a Gauss-Jordan elimination in order to attempt to diagonalize the leftmost part of the matrix. Upon success, we can conclude that we have a valid anchor; upon failure, we have an over-constraining anchor. The pseudo-code for this decision method is given at Algorithm 3.

The time complexity of this algorithm is that of the Gauss-Jordan elimination method, *i.e.* $\mathcal{O}(\min(m,n)mn)$. Since we are sure that the system is not generically over-constrained, $n \geq m$, the time complexity is $\mathcal{O}(m^2 n)$.

5. Detection of maximal well-constrained subsystems in articulated systems

In Section 4, we have addressed the issue of articulated systems, *i.e.* GCS which are under-constrained even *modulo* global transformation groups (translations, rotations, scalings, and their combinations), by giving an algorithm to decide whether a subset of the coordinates of the unknowns forms an anchor of the system. In this section, we also address the handling of articulated systems by giving means of identifying maximal G-well-constrained subsystems for the different groups G mentioned above. We use this to compute a set of maximal G-well-constrained subsystems which form a cover of the GCS.

We begin in Section 5.1 by explaining the identification of maximal rigid subsystems (MRS), then extend this in Section 5.2 to the identification of maximal G-well-constrained subsystems (MGS) for other groups than the direct isometries and explain what are the necessary conditions on a transformation group G for our method to work. Finally, in Section 5.3, we provide a skeletonization algorithm based on the identification of MGS.

For the sake of simplicity, we consider 2D systems in the rest of this section and consider thus that a rigid system has 3 degrees of freedom. Nevertheless, our algorithms work exactly the same way in 3D.

5.1. Identification of maximal rigid subsystems

The basic idea of our MRS detection algorithm is to study which geometric elements are fixed when one anchors a reference for the direct isometries. As explained in Section 4, within the witness framework, anchoring a reference for the direct isometries consists in switching columns in the Jacobian matrix so as to put the three columns of the reference in the rightmost positions. Recall (see Equation 2) then that after performing a Gauss-Jordan elimination method, the left sub-matrix of the Jacobian matrix is diagonalized and, thus, the ith row of the equation $J\overrightarrow{V} = 0$ can be read as

$$v_i = -\alpha_{1,i} \times v_{a_{n-m}} - \cdots - \alpha_{n-m,i} \times v_{a_1}$$

With a rigid GCS in 2D, there are 3 v_{a_k} elements. For instance, Table 4 shows the reduced row echelon form of the Jacobian matrix at the witness for the GCS of Figure 8. Since this GCS is rigid (with the redundant constraint removed), three columns do not belong to the identity part of the matrix: they correspond to coordinates x_4, y_4 and y_3, which form a reference for the system. All other coordinates can be expressed in function of these three coordinates. For instance, the first line of the matrix must be interpreted as $\dot{x_1} - \frac{4}{5}\dot{y_3} - \dot{x_4} + \frac{4}{5}\dot{y_4} = 0$, *i.e.* $\dot{x_1} = \frac{4}{5}\dot{y_3} + \dot{x_4} - \frac{4}{5}\dot{y_4}$.

When the GCS is not rigid, the size of an anchor is higher than 3. There are then more than three columns at the right of the identity sub-matrix after performing a Gauss-Jordan elimination. Table 5 shows the reduced row echelon form of the Jacobian matrix at a witness for the GCS of Figure 14. Notice that columns y_2 and y_4 were moved to the right, since it would have been

Table 5: Reduced row echelon form of the Jacobian matrix at a witness for the GCS of Figure 14

	$\dot{x_1}$	$\dot{y_1}$	$\dot{x_2}$	$\dot{x_3}$	$\dot{y_3}$	$\dot{x_4}$	$\dot{x_5}$	$\dot{y_5}$	$\dot{x_6}$	$\dot{y_2}$	$\dot{y_4}$	$\dot{y_6}$	$\dot{x_7}$	$\dot{y_7}$
r'_1	1	0	0	0	0	0	0	0	0	$\frac{4}{3}$	$\frac{101}{18}$	$-\frac{181}{108}$	-1	$-\frac{473}{108}$
r'_2	0	1	0	0	0	0	0	0	0	$-\frac{7}{3}$	$-\frac{40}{9}$	$\frac{28}{27}$	0	$\frac{140}{27}$
r'_3	0	0	1	0	0	0	0	0	0	4	$\frac{29}{2}$	$-\frac{15}{4}$	-1	$-\frac{59}{4}$
r'_4	0	0	0	1	0	0	0	0	0	0	$\frac{5}{2}$	$-\frac{17}{12}$	-1	$-\frac{37}{12}$
r'_5	0	0	0	0	1	0	0	0	0	0	$\frac{5}{2}$	$-\frac{7}{12}$	0	$-\frac{35}{12}$
r'_6	0	0	0	0	0	1	0	0	0	0	3	$-\frac{1}{6}$	-1	$-\frac{11}{6}$
r'_7	0	0	0	0	0	0	1	0	0	0	0	$-\frac{2}{3}$	-1	$\frac{2}{3}$
r'_8	0	0	0	0	0	0	0	1	0	0	0	$-\frac{1}{6}$	0	$-\frac{5}{6}$
r'_9	0	0	0	0	0	0	0	0	1	0	0	$-\frac{1}{6}$	-1	$\frac{7}{6}$

impossible to find a pivot and finish the Gauss-Jordan elimination otherwise. The variations of all coordinates can be expressed as functions of $\dot{y_2}$, $\dot{y_4}$, $\dot{y_6}$, $\dot{x_7}$ and $\dot{y_7}$. Indeed, an anchor for this GCS can consist in point p_7, direction p_7-p_6, direction p_5-p_4 and direction p_3-p_2.

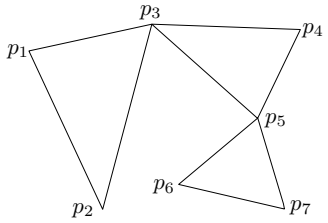

Figure 14: 2D articulated chain made of three rigid triangles. Distance constraints are implicitly represented by the segments.

An important result to identify MRSs comes from the zeros in columns $\dot{y_2}$ and $\dot{y_4}$. Rows 7, 8 and 9 of Table 5 can be interpreted as the fact that the values of $\dot{x_5}$, $\dot{y_5}$ and $\dot{x_6}$ depend only on those of $\dot{y_6}$, $\dot{x_7}$ and $\dot{y_7}$. Put differently, if one anchors a reference for the direct isometries by pinning down p_7 and direction p_7-p_6, then points p_6 and p_5 are fixed, *i.e.* $p_5p_6p_7$ is a rigid subsystem.

With this result, it is possible to design an algorithm to identify a set of MRSs which form a cover of the GCS. Such a cover is unique: if it were not, it would mean that there exists a rigid subsystem $\mathcal{S}$ which, in one of the existing covers, is split into two different MRSs. By definition of a MRS, if a part of $\mathcal{S}$ is included in a MRS, the other part also is.

A naive algorithm immediately arises, based on anchoring a reference for the direct isometries, *i.e.* switching columns to have the corresponding columns on the right of the Jacobian matrix and identifying the parts of the GCS which are fixed. The pseudo-code is shown at Algorithm 4. In this algorithm, anchoring a reference for the direct isometries means switching columns so as to have the columns corresponding to the reference at the right of the Jacobian matrix. In order to not identify the same MRS twice, we anchor references only on untagged parts of the GCS, that means that at least one of the columns cannot be tagged.

The cost of this algorithm depends on the number k of MRSs: for each of

Algorithm 4: Naive MRS identification algorithm

Input:
$\mathcal{S} = (C, X, A)$: a geometric constraint system
$\mathcal{W}$: a typical witness of $\mathcal{S}$
Result: Set of MRSs of $\mathcal{S}$

```
M ← ∅        // set of MRSs
J ← Jacobian matrix of S at W
i ← 0
repeat
    Anchor a reference for the direct isometries on an untagged part of S
    Perform a Gauss-Jordan elimination
    Tag with label i the columns of J which correspond to coordinates
    depending only on the 3 last columns
    M ← M ∪ {subsystem corresponding to the columns with tag i}
10  i ← i + 1
until all the columns are tagged
foreach constraint c ∈ C do
    if there is no system in M which includes c then
        M ← M ∪ {subsystem corresponding to c}
return M
```

them, it performs a Gauss-Jordan elimination only once, so that the total cost is $\mathcal{O}(k \min(n, m)nm)$, that is $\mathcal{O}(km^2n)$ since the system is not over-constrained. This cost can be reduced to $\mathcal{O}((k + m^2n)$, *i.e.* $\mathcal{O}(m^2n)$, by not starting the Gauss-Jordan elimination from scratch for each MRS. At the end of line 10 in the algorithm, the Jacobian matrix at the witness is in reduced row echelon form. By switching the columns in an appropriate way, one only needs to perform the Gauss-Jordan pivot operation on two to three columns. Indeed, by looking at the constraint graph, it is possible to select a new anchor for the GCS which satisfies the following conditions:

- it includes a reference for the direct isometries which is not totally tagged,
- each identified MRS is fixed, *i.e.*
  - the reference includes three coordinates in the MRS,
  - or the MRS shares a geometric element with a fixed MRS and the reference includes a coordinate in the MRS.

To select this reference, one only needs to consider a geometric element which is in an already identified MRS and which is linked by a constraint to an untagged element. More cases occur with systems for which the constraint graph has several connected components or with systems with implicit points (*e.g.* similarity-invariant systems with only lines and angles), but the principle remains the same. Thus, in most cases, one only needs to switch two columns, so

as to change the point in the reference. Three switches happen with disconnected graphs. Algorithm 5 shows how to perform MRS identification. For the sake of simplicity, the algorithm is described for articulated GCS made of several MRSs connected by points, but it is easily extended to systems with other kinds of geometric elements.

Algorithm 5: MRS identification algorithm for an articulated system

Input:
$\mathcal{S} = (C, X, A)$: a geometric constraint system
$\mathcal{W}$: a typical witness of $\mathcal{S}$
Result: Set of MRSs of $\mathcal{S}$

$J \leftarrow$ Jacobian matrix of $\mathcal{S}$ at $\mathcal{W}$
Anchor a reference for the direct isometries and identify and tag a first MRS
repeat
  Select a tagged point linked by a constraint to an untagged element
  Switch the columns of this point with the columns of the point in the last reference
  Perform Gauss-Jordan elimination on the two latter in order to identify a new MRS
  Tag the new MRS
until *all the columns are tagged*
foreach *constraint* $c \in C$ **do**
  if *there is no system in* M *which includes* c **then**
    $M \leftarrow M \cup \{$subsystem corresponding to $c\}$
return M

In the case of open chains, *i.e.* GCS where all cycles in the constraint graph are included in rigid subsystems, an even less costly algorithm exists, by using both the constraint graph and the Jacobian matrix. After performing the Gauss-Jordan elimination, a first MRS is identified by considering all the coordinates which depend only on the reference. From there, one can consider all the coordinates which depend on the reference and on one additional parameter. In the matrix of Table 5, with the additional parameter $\dot{y_4}$, we can fix $\dot{x_3}$, $\dot{y_3}$ and $\dot{x_4}$. Taking a look at the constraint graph, we notice that the previously identified MRS ($p_5p_6p_7$) shares only one point with the rest of the system and thus cannot "transfer" more than two degrees of freedom.

This enables us to remove the MRS and exchange the three parameters $\dot{y_6}$, $\dot{x_7}$ and $\dot{y_7}$ with parameters $\dot{x_5}$ and $\dot{y_5}$ in the Jacobian matrix. The numerical values are not important in this process: we consider that all the values of both columns are non-zero. With this new matrix, one notices that x_5, y_5 and y_4 form a reference for the direct isometries and that by anchoring the variations of this reference, $\dot{x_3}$, $\dot{y_3}$ and $\dot{x_4}$ are fixed, *i.e.* $p_3p_4p_5$ is a rigid system. We continue this algorithm by noticing that this system shares only one point with

the rest of the system, removing it and replacing it with non-zero-filled columns $\dot{x_3}$ and $\dot{y_3}$ and thus identifying the last MRS $p_1p_2p_3$.

When the last identified MRS shares more than one point with the rest of the system, two cases occur: either the removal of the MRS leads to two disconnected graphs (*i.e.* the MRS is in the middle of the articulated system) and one thus continues the algorithm separately on each part of the graph; or the MRS belongs to a non-rigid closed chain.

When one uses this algorithm on a GCS containing non-rigid closed chains, it leads to cases where one cannot detect the MRSs of the closed chains, because of the inter-dependence of the rigid subsystems of the chain. After identifying the first MRS of the closed chain, the algorithm is stuck because it is not possible to identify another system which depends only on three parameters. In this case, we get back to Algorithm 5 to identify the different MRSs of the closed chain.

5.2. Extension to other transformation groups

Algorithm 4 can be adapted to identify maximal G-well-constrained subsystems, for transformation groups G other than the direct isometries (see [37] or [29] for formal definitions of the transformation groups in the context of geometric constraints).

There are several conditions on G. First of all, if one wants to check that the whole system is G-invariant, in order to know if the system is under-constrained *modulo* G, one needs to be able to simulate the action of G in the Jacobian matrix. In [30], we showed how to simulate the action of the translations and rotations, which enables the recognition of these groups in the kernel of the Jacobian matrix. As mentioned in Section 4, it is possible to likewise simulate the action of scalings. In order to simulate a scaling centered on point O and with a scaling factor of f, we apply a translation on a point p_1, with direction $\overrightarrow{Op_1}$ and a norm x, and another translation on a point p_2, with direction $\overrightarrow{Op_2}$ and a norm $\frac{||\overrightarrow{Op_2}||}{||\overrightarrow{Op_1}||}x$.

Second, one needs to be able to anchor a G-reference, which means one must be able to select columns of the Jacobian matrix which correspond to a G-reference. We already described how to do this for the direct isometries. In order to anchor a reference for the similarities, for instance, one needs to consider as fixed the coordinates of two points in 2D, with an additional coordinate of a third point (in order to simulate the anchoring of the directions between the two fixed points and the third point) in 3D.

With these two conditions fulfilled, adapting Algorithm 4 to identify maximal G-well-constrained subsystems (MGS) is straightforward: instead of anchoring references for the direct isometries, one anchors G-references. Beforehand, one only needs to remove all non-G-invariant constraints[5]. The pseudo-code is given at Algorithm 6.

[5] In some cases – when parameter values are not independent, or when using exotic formulations (*e.g.* Cayley-Menger determinants to specify colinearities, cocyclicities and coplanarities) – removing metric constraints can lead to geometric constraint systems which are projectively

Since we use the genericity hypothesis, we consider here that the parameters are independent. Otherwise, removing the G-invariant constraints might lead to losing information: for instance, if two constraints have the same metric parameters, they induce an equality constraint, but not under the genericity hypothesis, since using other values of the parameters removes this equality.

Algorithm 6: Naive MGS identification algorithm

Input:
$\mathcal{S} = (C, X, A)$: a geometric constraint system
$\mathcal{W}$: a typical witness of $\mathcal{S}$
G: a transformation group
Result: Set of maximal G-well constrained subsystems of $\mathcal{S}$

```
M ← ∅          // set of MGSs
J ← Jacobian matrix of S at W
i ← 0
foreach constraint c ∈ C do
    if c is not a G-invariant constraint then
        C ← C/{c}
repeat
    Anchor a G-reference on an untagged part of S
    Perform a Gauss-Jordan elimination
    Tag with label i the columns of J which correspond to coordinates
    depending only on the columns of the G-reference
    M ← M ∪ {subsystem corresponding to the columns with tag i}
    i ← i + 1
until all the columns are tagged
foreach constraint c ∈ C do
    if there is no system in M which includes c then
        Anchor a G-reference including c
        Identify the corresponding MGS m_c
        M ← M ∪ {m_c}
return M
```

It is possible, as was the case for Algorithm 4, to adapt this algorithm in order to perform fewer steps of the Gauss-Jordan elimination, by taking into account the constraint graph and reducing the number of column switches between two MGS.

Also, notice that it is possible to consider a rigid system and to make it well-constrained *modulo* the similarities, by replacing the unit of distance constraints with a parameter. This is useful in industrial CAD applications, where

less constrained than the initial system. For the sake of conciseness and simplicity, we ignore those cases.

previously solved systems are re-used at a different scale. Furthermore, in such a case, a witness is directly available: a solution of the previously solved system.

5.3. Skeletonization of a geometric constraint system

In this section, we propose an algorithm to skeletonize a geometric constraint system, that is to say transform a system $\mathcal{S}$ into a system $\mathcal{S}'$ such that $|\mathcal{F}(\mathcal{S})| = |\mathcal{F}(\mathcal{S}')|$ and such that any figure in $\mathcal{F}(\mathcal{S}')$ is a subfigure of a figure in $\mathcal{F}(\mathcal{S})$. The basic idea is to replace maximal G-well-constrained subsystems with their boundary system.

Skeletonization algorithms already exist in geometric constraint solving: graph-based retropropagation algorithms [5, 32, 41] remove geometric elements and constraints that can be built if the rest of the system is built, until they get a system which needs to be built from scratch. This remaining system, the skeleton, has as many degrees of freedom as the initial system, and any variation of the coordinates of an element in the initial system can be expressed as a function of the variation of the coordinates of the skeleton elements.

The interest of skeletonization also lies in the graphical feedback it gives to the user about the flexibility of the system: for instance, if the skeleton of the GCS is made of two bars linked by a common point, the user instantly sees that the GCS is made of two rigid systems which can rotate around their common link.

In order to transform a GCS into its skeleton, we use Algorithm 6, in order to identify maximal G-well-constrained subsystems, and Algorithm 2, in order to compute the boundary of the identified MGSs. The algorithm consists in identifying all MGSs and replaces them with their boundary. The pseudo-code is given at Algorithm 7.

Algorithm 7: G-skeletonization of a GCS

Input:
$\mathcal{S}$: a geometric constraint system
$\mathcal{W}$: a typical witness of $\mathcal{S}$
G: a transformation group
Result: G-skeleton of $\mathcal{S}$

$J \leftarrow$ Jacobian matrix of $\mathcal{S}$ at $\mathcal{W}$
$M \leftarrow$ list of MGSs of $\mathcal{S}$ identified by Algorithm 6
foreach *MGS* $k = (C_k, X_k, A_k) \in M$ **do**
    $B = (C_b, X_b, A_b) \leftarrow$ boundary system of k computed with Algorithm 2
    if $X_b \neq X_k$ **then**
        Replace k with B

The time complexity of Algorithm 7 is as follows. It uses Algorithm 6, which is in time $\mathcal{O}(m^2n)$. For each MGS of the system, it uses Algorithm 2, which is also in $\mathcal{O}(m^2n)$. If there are k MGSs, the overall time complexity is thus in $\mathcal{O}(km^2n)$.

6. $\mathcal{W}$-decomposition of a GCS

The previous section gives algorithms to identify all MGSs of a GCS. Having such an algorithm leads to a natural method to decompose a G-well-constrained geometric constraint system. We call this method $\mathcal{W}$-decomposition and a system which can be decomposed by this method is said to be $\mathcal{W}$-decomposable. In this section, we explain the principles of $\mathcal{W}$-decomposition and give examples.

Algorithm 6 identifies maximal G-well-constrained subsystems, *i.e.* if a MGS can be decomposed in several G-well-constrained subsystems, this will not be detected. The basic idea of $\mathcal{W}$-decomposition is to remove constraints from the system and see if it breaks the MGS in non-trivial MGSs, *i.e.* MGSs which are not limited to their boundary (*e.g.* a system limited to a point-point distance). If it does, then we use $\mathcal{W}$-decomposition on each non-trivial MGS. Algorithm 8 gives the pseudo-code of the algorithm.

Algorithm 8: $\mathcal{W}$-decomposition

Input:
  $\mathcal{S} = (C, X, A)$: a G-well-constrained geometric constraint system
  $\mathcal{W}$: a typical witness of $\mathcal{S}$
Result: Tree of G-well-constrained subsystems of $\mathcal{S}$
repeat
2 Select a constraint $c \in C$
3 $L \leftarrow$ list of the MGSs of $(C \backslash c, X, A)$ identified using Algorithm 6
  while L *contains only trivial MGSs* **do**
    Select a constraint c which was not selected yet
    $L \leftarrow$ list of the MGSs of $(C \backslash c, X, A)$ identified using Algorithm 6
until *all constraints have been tested or we find a non-trivial MGS*
if L *contains only trivial MGSs* **then**
  return *a leaf labeled with* $\mathcal{S}$
else
  $\mathcal{A} \leftarrow$ childless node labeled with $\mathcal{S}$
  foreach $\mathcal{S}_i \in L$ **do**
13 Root the $\mathcal{W}$-decomposition of $\mathcal{S}_i$ as child of $\mathcal{A}$
    $\mathcal{S} \leftarrow \mathcal{S} - \mathcal{S}_i$
    $\mathcal{S} \leftarrow \mathcal{S}+$ boundary of $\mathcal{S}_i$
16 Root the $\mathcal{W}$-decomposition of $\mathcal{S}$ as child of $\mathcal{A}$
  return $\mathcal{A}$

Let us illustrate this algorithm on the example of Figure 15a, which represents the constraint graph of a 2D rigid GCS. The graph is 3-connected and has two $K_{3,3}$ subgraphs[6], connected by three "middle" edges (p_1p_2, p_3p_4 and

[6] The $K_{3,3}$ structure of the subgraphs is not obvious, but is better seen by numbering the vertices clockwise and considering the subsets of even vertices and odd vertices.

p_5p_6). Algorithm 6 detects the rigidity of the whole system. Let us consider the removal of two constraints at line 2 of Algorithm 8: dotted edges e_1 and e_2.

If we remove edge e_1, the use of Algorithm 6 at line 3 identifies four MGSs: the rigid $K_{3,3}$ subsystems, and each edge between them. The latter are equivalent to their boundary. Replacing the rigid hexagons by their boundaries and reintroducing edge e_1 leads to the graph of Figure 15b (note that edge e_1 must be taken into account for the computation of the boundaries). The recursive use of $\mathcal{W}$-decomposition (line 13) on each non-trivial MGS leads to the knowledge that they are not $\mathcal{W}$-decomposable. The same happens with the recursive use on the system of Figure 15b (line 16).

If we do not remove edge e_1 but e_2 instead, the left $K_{3,3}$ subsystem of Figure 15a is no longer rigid. The identification of non-trivial MGS thus only identifies the hexagon on the right of Figure 15a. Once it is replaced by its boundary, we obtain the system shown on Figure 15c. The recursive use of $\mathcal{W}$-decomposition will then lead, after removal of one of the three "middle" edges, to the identification of the second rigid hexagon and thus to the system shown on Figure 15b.

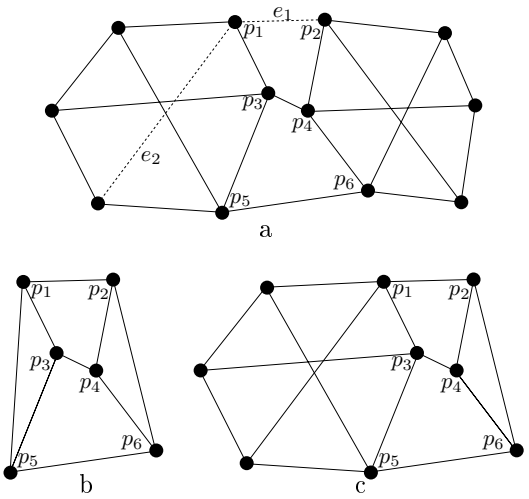

Figure 15: 2D systems where edges represent point-point distances; (a): 3-connected constraint graph made of two $K_{3,3}$ graphs connected with 3 constraints; (b) and (c): graphs obtained by replacing MRSs identified by Algorithm 8 by their boundary with respectively edges e_1 and e_2 removed.

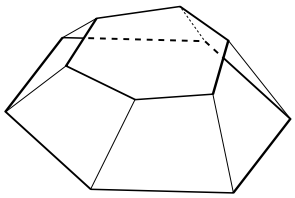

Figure 16: Generalized Stewart platform: both thick hexagons are rigid; the other segments indicate distance constraints.

Let us consider the 3D example of Figure 16, a generalized Stewart platform [40]. It consists of two platforms (the rigid hexagons in thick lines) linked by six distance constraints. The whole system is rigid, which is easily detected by the witness method (it is also characterized as rigid by the 3D extension of Laman's theorem [24]). Since any rigid subsystem can be identified by Algorithm 6, no matter how it is composed, the $\mathcal{W}$-decomposition will identify:

- both platforms as rigid if one of the six distance constraints is selected;
- one platform as rigid if selection of a constraint in the other platform makes it articulated.

Of course, whatever constraint is selected first, both platforms will eventually be identified as rigid. Notice that whatever rigid subsystem is linked to the platforms, it will be identified as rigid and replaced by its boundary, *i.e.* a minimal rigid hexagon.

Execution time depends on the choice of the removed constraint. In the worst case, all constraints are tested: m times the Algorithm 6 is used, thus the time complexity is $\mathcal{O}(m^3 n)$.

Our algorithm is more powerful than algorithms found in the literature, for several reasons:

- first of all, it is independent of the connectivity of the constraint graph. For instance, Figure 17a gives an example of a 4-connected constraint graph which is $\mathcal{W}$-decomposable, no matter what is inside the inner blue part as long as it is rigid; we may likewise build $\mathcal{W}$-decomposable systems with a k-connected constraint graph, for any k, by considering two polygons with k vertices, linked by k constraints, one of the polygons being rigid (see Figure 17b);
- second, it is also not based on a cluster formation. Since the graph of Figure 17c is not decomposable by current graph decomposition methods, the system of Figure 17a, with the inner part replaced by Figure 17c, will also lead to a decomposition failure for these methods, whereas it is $\mathcal{W}$-decomposable.

Ultimate decomposition consists in yielding a triangular equation system. For algebraic systems, Ritt-Wu decomposition [1] or Gröbner bases with lexical order lead to such decompositions, but unfortunately, they are intractable in the CAD domain. On the other hand, $\mathcal{W}$-decomposition is not as powerful as these algebraic methods since it is possible to construct an infinite family of $\mathcal{W}$-indecomposable constraint systems like the one depicted in Figure 17d: there is no constraint in this system such that its removal produces a system with a MRS bigger than a point-point distance. But, on the positive side, it is easy to see that

- all Owen-decomposable systems [34] are $\mathcal{W}$-decomposable (that is, articulation pairs are detected by the choice of the deleted constraint)
- all constraint systems which are decomposable by cluster formation methods or on the search of minimal rigid parts, are also $\mathcal{W}$-decomposable.

We think that the ratio of efficiency to power of decomposition is good enough to give good results in CAD even in the 3D case.

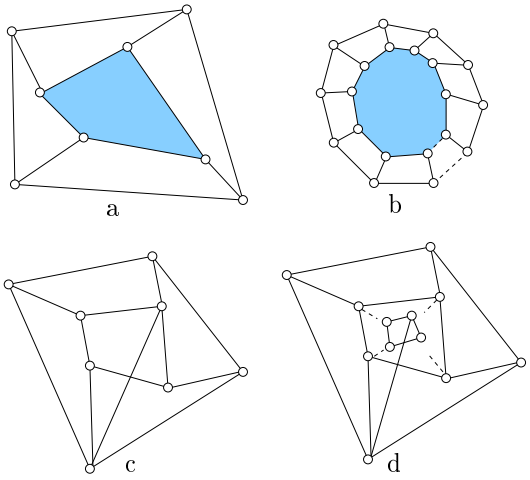

Figure 17: 2D rigid examples for the $\mathcal{W}$-decomposition: each vertex is a point and each edge represents a distance constraint. (a): $\mathcal{W}$-decomposable 4-connected GCS (the blue subsystem is rigid); (b): $\mathcal{W}$-decomposable k-connected GCS (the blue subsystem is rigid); (c): $\mathcal{W}$-indecomposable system; (d): there are $\mathcal{W}$-indecomposable systems with an arbitrary number of points.

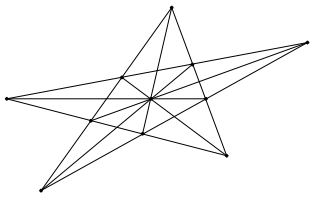

Figure 18: System with only point-line incidence constraints and without any rational solution

7. Robustness issues

7.1. The problem

Our method assumes that it is possible to compute the rank of a set of vectors, given by their coordinates. It is a basic problem in computerized linear algebra with well-known methods. Only at first glance does it seem to be an easy problem.

Since the rank is not a continuous function, it is not computable in the sense of Computable Analysis [43]. In short, Computable Analysis represents real numbers with sequences of nested intervals; the bounds of these intervals are long float numbers, typically represented as $m2^e, m \in \mathbb{Z}, e \in \mathbb{Z}$; the mantissas m are calculated with an arithmetic managing long integers. The width of intervals enclosing the real numbers can be made arbitrarily small (within the limitation of computing power and memory), but it is never zero. This arithmetic can handle rational numbers, algebraic numbers, and transcendental numbers.

It is semi-deterministic in the following sense. On the one hand, it can detect that a number is non-zero: compute a sufficiently precise interval which does not contain zero; on the other hand, it cannot detect that a number (*e.g.* a Gauss pivot, or a determinant) is zero when this number is null: it would require reaching the interval $[0, 0]$, which is impossible in finite time. For example, this arithmetic can compute intervals enclosing $\sqrt{2}$ with great accuracy, but intervals for $(\sqrt{2})^2 - 2$ will never be $[0, 0]$.

Similarly, this arithmetic can decide that two distinct real numbers are different, but cannot decide that two equal numbers are equal. A consequence is that it is possible to numerically prove that a set of given vectors are linearly independent (when they are), but it is not possible to numerically prove that they are dependent when they are.

If a rational witness is available, an exact rational arithmetic can be used. The rank of rational vectors is computable, and this approach is practical. It is explored in [30] with a number of examples.

Sometimes a rational witness is not available; it also happens that some problems have no rational solution, and thus no rational witness at all. Section 7.3 discusses which problems have rational solutions, and which have not. Notice that no numerical solver provides an exact rational witness, even when it exists: the solution returned by a numerical solver is either a floating-point approximation, or some small box containing an isolated root, or some small box of a cover of the solution manifold.

A theoretical solution is to resort to some exact algebraic arithmetic, when the constraint system is algebraic. However, this approach is not practical and its relevance is questionable. We here present a more reasonable solution.

7.2. Numerically typical witnesses

The simplest and radical solution to this problem is to require the witness to be not only typical, but also to be *numerically typical*. For example, for three non-collinear points A, B, C, the angle between AB and BC must be sufficiently different from π and from 0 in a numerically typical witness. Then it suffices to use the classical epsilon heuristic: if the angle between AB and BC is close to π or 0, *i.e.* the 3 points A, B, C are numerically very close to alignment, then they indeed are collinear, and the difference from 0 or π is due to numerical inaccuracy.

This requirement seems to be a reasonable prerequisite for a witness. All dynamical geometry softwares (Cabri Géomètre, Cinderella, GeoGebra, *etc.*) which are used for geometry teaching already rely on this requirement, and use it routinely: the student or the teacher interactively specifies some geometric construction (say with ruler and compass), then interrogates the dynamical geometry software to know if some property (alignment of three points, cocyclicity of four points) holds. The figure is considered as a numerically typical witness, and the request is answered considering the coordinates of points, lines and circles in the figure.

Luckily, the context of geometric constraint solving is a situation where the epsilon heuristic is sufficient: in Computational Geometry, where people cannot choose their data, the epsilon heuristic is not sufficient to achieve robustness.

7.3. Geometric problems and rational realizations

Some geometric problems have no rational solution, thus no rational witness, for instance a regular pentagon. More surprisingly, it is possible to build

2D problems which involve only point-line incidences and which have no rational solution. For instance the pentagon in Figure 18 involves no metric constraints at all (neither distance nor angle are specified) but it is projectively equivalent to the regular pentagon, or in other words, it is a perspective of a regular pentagon; this pentagonal star is not realizable in the rationals. Almost all regular n-polygons, as well as their projective variant which involves only point-line incidences, are non-realizable with rationals, exceptions being $n = 3, 4, 6$; for instance an equilateral triangle has the rational realization: $(1, 0, 0), (0, 1, 0), (0, 0, 1)$ and this extends easily to regular hexagons. The latter "cheating" trick (using 3D coordinates for a 2D problem) cannot be used for pentagons, heptagons, *etc.* but the proof is omitted for conciseness.

To sum up, some 2D problems involving only point-line incidences have no rational solution, and thus no rational witness. Of course, if such a system is a subsystem of the system to be solved, whether there are other types of constraints or not, the absence of rational solutions and witnesses holds.

Other less artificial 2D geometric problems without rational solution involve metric constraints, for instance bisecting lines (*i.e.* equal angles): the bisecting line of two rational vectors is generally non-rational, because some square root is involved to solve the underlying quadratic equation. In contrast, the bisector of a segment joining two rational points does not require a square root and is a rational construction.

Let us now consider 3D problems. As shown by Steinitz's theorem [36], every 3D Eulerian polyhedron (*i.e.* fulfilling Euler-Poincaré's formula: $V - E + F = 2$, where V, E, F are the numbers of vertices, edges and faces) is realizable with a 3D convex polyhedron with rational coordinates only (thus with integer coordinates only, after some scaling). A cubic time algorithm computes a Tutte barycentric embedding of the planar graph[7] of the polyhedron, then lifts vertices in 3D [36]. In contrast, 4D polytopes (convex polyhedra) are generally not realizable with integer coordinates only [36]. Less is known for 3D non-Eulerian polyhedra (*i.e.* with through holes, such as a torus). However, in spite of their intellectual appeal and of the fact that some of them have a known complexity, these problems are seldom seen in CAD/CAM designs. Thus, the impossibility to handle them is only a meager shortcoming of our method.

8. Conclusion

After proposing a way to generate a witness, we showed in this paper how the witness method could be used to detect over-constrained systems without any additional computational cost by an incremental Gauss-Jordan elimination of the Jacobian matrix at the witness. This allows the computation of a well-constrained boundary inside the decomposition method.

We propose algorithms to validate anchors, *i.e.* check that fixing the position of a subset of the coordinates of the geometric elements fixes all coordinates. We

[7] Each vertex is the barycenter of its neighbors, except for three outwards base vertices.

gave algorithms to identify all maximal well-constrained subsystems of a GCS, *i.e.* the system itself if it is well-constrained, or its G-well-constrained parts if it is articulated.

From this algorithm, we deduced a skeletonization algorithm, allowing to compute a minimized version of the GCS, with the same degrees of freedom, based on the replacement of maximal G-well-constrained subsystems with their boundary. We also deduced a method, called $\mathcal{W}$-decomposition, to decompose a G-well-constrained GCS into the set of all its non-trivial G-well-constrained subsystems, based on the removal of a constraint and the computation of the new maximal G-well-constrained subsystems.

The method to detect over-constrainedness is efficient (the computation of the reduced row echelon form of the Jacobian matrix is in $\mathcal{O}(\min(m,n)mn)$) and is not tricked by mathematical theorems, even when these theorems are unknown to the developer. The decision algorithm for the validity of an anchor is in $\mathcal{O}(m^2n)$. The MGS identification is also efficient ($\mathcal{O}(m^2n)$) and works in 2D or 3D. $\mathcal{W}$-decomposition is performed in $\mathcal{O}(m^3n)$ in the worst case.

For conciseness reasons, the algorithms we described work on 2D systems, but they can be easily extended to 3D systems. The main difficulty to implement our algorithms is the computation of the Jacobian matrix and its manipulation, but there are many development libraries which can do it both efficiently and robustly. Our algorithms are thus easy to implement. Complexity of the algorithms is independent of the dimension. Even though this complexity is cubic, we believe that it is not an important drawback, thanks to the incrementality of our algorithms, allowing the developer to use idle user time for computations.

As it is, the method has been implemented and tested on several examples. However, some work could be done in order to improve its efficiency and its robustness. For instance, the infinitesimal flexions are discovered by computing the kernel of $F'(X,A)$ on a witness, this calculus can be confirmed by using the Hessian matrix of each equation of the system. The nullity of the product $\overrightarrow{V}^t H(X,A)\overrightarrow{V}$ should confirm or infirm this fact: contradiction would mean that our witness is not generic and small perturbations could be used to remedy this situation.

Since reliability of the witness is a crucial point in our method, the previous point can re-reinforce the confidence we have in the witness. Some other tracks can be explored, like computing a witness fulfilling exactly the boolean constraints (incidence constraints, equality of distances, *etc.*): that is solving constraint systems *modulo* the group of the projective transformations.

Another different approach could consist in turning the parameters into variables and apply the method as it is. That way, the relationships between parameters could be detected and the flexions could be quantitatively estimated; this could be useful for engineering studies.

Further research needs to be done in order to have efficient $\mathcal{W}$-decomposition. The example of Figure 15 shows that some edges are better than others for the removal (line 2 of Algorithm 8). Some promising tracks are the computation of a minimum chain covering and the search for constraints which appear in only

a few chains, or methods based upon matroids intersections.

Those enhancements of the $\mathcal{W}$-decomposition method would not modify the complexity of the algorithm. A multi-group $\mathcal{W}$-decomposition algorithm is more promising: Algorithm 8 works on any transformation group which fulfills the conditions cited in Section 5.2, but only one such group at a time. To establish a multi-group algorithm, *i.e.* an algorithm decomposing a system into its G_1-well-constrained subsystems, then automatically switching to another group G_2 when necessary, would enhance the resolution power of the algorithm. However, defining an interesting order between the transformation groups is not as straightforward as it may seem, and we intend to look into this.

References

[1] S.-C. Chou and X.-S. Gao. Ritt-Wu's decomposition algorithm and geometry theorem proving. In *CADE '90: Proceedings of the 10th International Conference on Automated Deduction*, pages 207–220, Kaiserslautern, Germany, 1990. Springer.

[2] J.-F. Dufourd, P. Mathis, and P. Schreck. Geometric construction by assembling solved subfigures. *Artificial Intelligence*, 99(1):73–119, 1998.

[3] C. Durand and C. Hoffmann. A systematic framework for solving geometric constraints analytically. *Journal of Symbolic Computation*, 30(5):493–529, 2000.

[4] C. Essert-Villard, P. Schreck, and J.-F. Dufourd. Sketch-based pruning of a solution space within a formal geometric constraint solver. *Artificial Intelligence*, 124(1):139–159, 2000.

[5] A. Fabre and P. Schreck. Combining symbolic and numerical solvers to simplify indecomposable systems solving. In *SAC '08: Proceedings of the 23rd ACM Symposium on Applied Computing*, pages 1838–1842, Fortaleza, Brazil, 2008. ACM.

[6] G. Ferro, G. Gallo, and R. Gennaro. Probabilistic verification of elementary geometry statements. In *ADG '96: First International Workshop on Automated Deduction in Geometry*, volume 1360 of *Lecture Notes in Artificial Intelligence*, pages 87–101, Toulouse, France, 1996. Springer.

[7] I. Fudos and C. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, 16(2):179–216, 1997.

[8] 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, California, USA, 2009.

[9] X.-S. Gao, C. Hoffmann, and W.-Q. Yang. Solving spatial basic geometric constraint configurations with locus intersection. *Computer-Aided Design*, 36(2):111–122, 2004.

[10] X.-S. Gao, Q. Lin, and G.-F. Zhang. A C-tree decomposition algorithm for 2D and 3D geometric constraint solving. *Computer-Aided Design*, 38(1):1–13, 2006.

[11] J. Graver, B. Servatius, and H. Servatius. *Combinatorial Rigidity*. Graduate Studies in Mathematics. American Mathematical Society, 1993.

[12] B. Hendrickson. Conditions for unique graph realizations. *SIAM Journal on Computing*, 21(1):65–84, 1992.

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

[14] C. Hoffmann, A. Lomonosov, and M. Sitharam. Finding solvable subsets of constraint graphs. In *CP 1997: Proceedings of the 3rd International Conference on Principles and Practice of Constraint Programming*, pages 463–477, Hagenberg Castle, Austria, 1997.

[15] C. Hoffmann, M. Sitharam, and B. Yuan. Making constraint solvers more usable: overconstraint problems. *Computer-Aided Design*, 36(4):377–399, 2004.

[16] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied Interval Analysis*. Springer, 2001.

[17] C. Jermann, B. Neveu, and G. Trombettoni. Algorithms for identifying rigid subsystems in geometric constraint systems. In *IJCAI '03: Proceedings of the 18th International Joint Conference on Aritificial Intelligence*, pages 233–238, Acapulco, Mexico, 2003. Morgan Kaufmann.

[18] 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.

[19] R. Joan-Arinyo and A. Soto-Riera. A correct rule-based geometric constraint solver. *Computer and Graphics*, 5(21):599–609, 1997.

[20] R. Joan-Arinyo and A. Soto-Riera. Combining constructive and equational geometric constraint-solving techniques. *ACM Transactions on Graphics*, 18(1):35–55, 1999.

[21] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana-Pasto. Revisiting decomposition analysis of geometric constraint graphs. *Computer-Aided Design*, 36(2):123–140, 2004.

[22] P. Jörg, M. Sitharam, Y. Zhou, and J. Fan. Elimination in generically rigid 3D geometric constraint systems. In *Algebraic Geometry and Geometric Modeling*, Mathematics and Visualization, pages 205–216, Nice, France, 2004. Springer-Verlag.

[23] R. Kearfott and M. Novoa. INTBIS, a portable interval Newton/bisection package. *ACM Transactions on Mathematical Software*, 16(2):152–157, 1990.

[24] G. Laman. On graphs and rigidity of plane skeletal structures. *Journal of Engineering Mathematics*, 4(4):331–340, 1970.

[25] H. Lamure and D. Michelucci. Solving geometric constraints by homotopy. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):28–34, 1996.

[26] M. Larive, O. Le Roux, and V. Gaildrat. Using meta-heuristics for constraint-based 3D objects layout. In *3IA '04: Proceedings of the International Conference on Computer Graphics and Artificial Intelligence*, pages 11–23, Limoges, France, 2004.

[27] R. Latham and A. Middleditch. Connectivity analysis: a tool for processing geometric constraints. *Computer-Aided Design*, 28(11):917–928, 1996.

[28] A. Mantler and J. Snoeyink. Banana spiders: a study of connectivity in 3D combinatorial rigidity. In *CCCG '04: Proceedings of the 16th Canadian Conference on Computational Geometry*, pages 44–47, Montréal, Québec, Canada, 2004.

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

[30] 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, California, USA, 2009. ACM.

[31] D. Michelucci, S. Foufou, L. Lamarque, and D. Ménegaux. Another paradigm for geometric constraints solving. In *CCCG '06: Proceedings of the 18th Annual Canadian Conference on Computational Geometry*, pages 169–172, Queen's University, Ontario, Canada, 2006.

[32] A. Noort, M. Dohmen, and W. Bronsvoort. Solving over- and underconstrained geometric models. In *Geometric Constraint Solving and Applications*, chapter 2, pages 107–127. Springer, 1998.

[33] J.-J. Oung, M. Sitharam, B. Moro, and A. Arbree. FRONTIER: fully enabling geometric constraints for feature-based modeling and assembly. In *SMA '01: Proceedings of the 6th ACM symposium on Solid Modeling and Applications*, pages 307–308, Ann Arbor, Michigan, USA, 2001. ACM.

[34] J. Owen. Algebraic solution for geometry from dimensional constraints. In *SMA '91: Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*, pages 397–407, Austin, Texas, United States, 1991. ACM.

[35] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, 1992.

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

[37] P. Schreck and P. Mathis. Geometrical constraint system decomposition: a multi-group approach. *International Journal of Computational Geometry and Applications*, 16(5,6):431–442, 2006.

[38] M. Sitharam. Well-formed systems of point incidences for resolving collections of rigid bodies. *International Journal of Computational Geometry and Applications*, 16(5,6):591–615, 2006.

[39] A. Sommese and C. Wampler. *The Numerical Solution of Systems of Polynomials arising in Engineering and Science*. World Scientific, 2005.

[40] D. Stewart. A platform with six degrees of freedom : A new form of mechanical linkage which enables a platform to move simultaneously in all six degrees of freedom developed by Elliott-automation. *Aircraft Engineering and Aerospace Technology*, 38(4):30–35, 1966.

[41] G. Trombettoni and M. Wilczkowiak. GPDOF: a fast algorithm to decompose under-constrained geometric constraints: Application to 3D modeling. *International Journal of Computational Geometry and Applications*, 16(5-6):479–511, 2006.

[42] H. van der Meiden and W. Bronsvoort. A non-rigid cluster rewriting approach to solve systems of 3D geometric constraints. *Computer-Aided Design*, 42(1):36–49, 2010.

[43] K. Weihrauch. *Computable analysis: an introduction*. Springer, 2000.

[44] L. Yang. Solving geometric constraints with distance-based global coordinate system. In *International Workshop on Geometric Constraint Solving*, Beijing, 2003.

[45] C. Yap. *Fundamental problems in algorithmic algebra*. Oxford University Press, 2000.

[46] G.-F. Zhang and X.-S. Gao. Well-constrained completion and decomposition for under-constrained geometric constraint problems. *International Journal of Computational Geometry and Applications*, 16(5,6):18–35, 2006.