Why are under-constrained systems not that bad

Simon Thierry, Pascal Schreck, Pascal Mathis

Why under-constrained systems are not that bad

Simon E.B. Thierry, Pascal Schreck, and Pascal Mathis

LSIIT, UMR CNRS
Université de Strasbourg
[simon.thierry,schreck,mathis]@unistra.fr

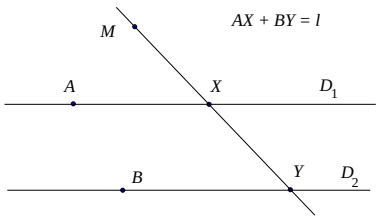
Abstract. Under-constrained geometric constraint systems are often considered as mis-constrained systems which have to be corrected by a completion mechanism. We expose here some works performed in our team and where under-constrained systems are considered as a wish of the designer or a step used in order to solve a well-constrained system.

1 Introduction

The main goal of the geometric constraint solving problem consists in yielding objects declaratively specified by the means of both a geometric description involving the characteristic entities of the object, like points, lines, planes or circles, and the relations between these entities, also named geometric constraints.

The problematic of geometric constraint solving mainly arises in two fields of computer science. The first one is the Computer Aided Education domain (CAE), where problems from high school mathematical programs like the following one (see "Statement" below) have to be considered in the context of dynamic geometry ([1, 2]) or Computer Assisted Proof in geometry ([3–6]).

Statement. Let D_1 and D_2 be two given lines, A be a point on D_1, B be a point on D_2 and M any point. Construct a line d passing through M and crossing D_1 in point X and D_2 in point Y such that distance $AX + BY$ is equal to a given constant l (see the figure alongside).



Although many features of this domain are interesting in order to have a good understanding of the geometric constraint solving problematic, the focus will not be put on this domain in this paper [7, 8].

The second domain where geometric constraint solving has been more widely studied is the Computer Aided Design and Drawing field (CAD) (see for instance [9–16]). In this framework, the geometric entities and relations are given under the form of a sketch on which the user imposes a dimensioning (*cf.* figure 1). With the progress of computer science and the advent of CAD, technical design softwares enriched with functionalities that automatically solve this kind of problems:

– the user draws a sketch using the graphical interface of the software,
– he/she then imposes a dimensioning with specific tools,
– a software module (called a *solver*) modifies the initial drawing so that it satisfies the dimensioning.

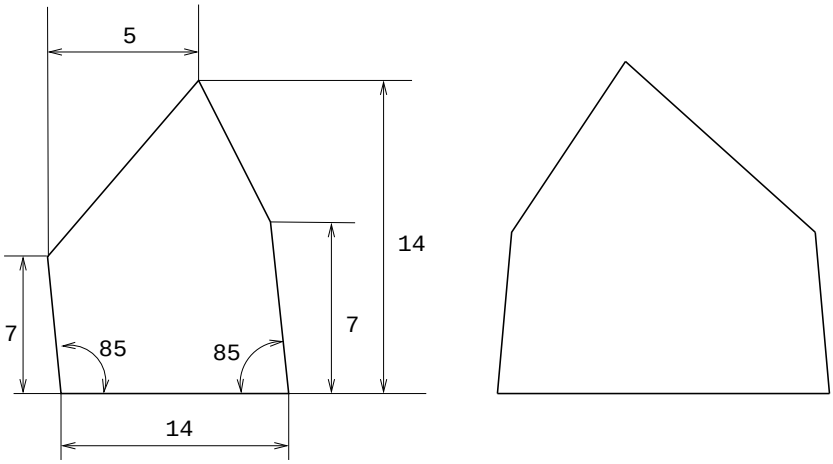


Fig. 1. A technical sketch (left) and a scaled solution (right) without its dimensioning. The dimensioning is normalized and depicted by the arrows: straight arrows mean distance constraints and curved arrows mean angular constraints.

The main differences between CAE and CAD contexts lie in the form of the statements (literal *vs* pictorial) and the expected nature of their solutions:

– in CAE, one wants a *way* to construct *all* the solutions, *i.e.* a *program* or a *macro* whose input is the position of the given entities and whose output is the solutions,
– in CAD, the user only needs one drawing meeting the metric requirements and close to the shape of the sketch.

The question of being able to yield all the solutions, even in CAD, is not as meaningless as it seems at first sight. Indeed, it is difficult, and cumbersome, to write down a non-ambiguous geometric specification describing a single object and dimensioned sketches thus often define more than one object. For instance, the specification of a triangle by the lengths of its three sides, generally defines two triangles up to a displacement (also called a direct isometry or a rigid body motion). In other words, when a vertex and an edge of the triangle are fixed, there are two solutions for this particular geometric constraint system. With n distance constraints between $2n-3$ points, there are 2^{n-2} solutions. In the CAD context, the solver should be able to select the more promising solution and/or to provide a way to smartly browse the set of the solutions (called *solution space*).

A constraint system with a finite number of solutions is said well-constrained and various studies have yet been done to select one solution and to browse the solution space [12, 17–20]. A constraint system with an infinite number of solutions is said under-constrained, and this state has often been viewed as a negative fact that the solver should detect and correct [21, 22]. In fact, things are a little bit more complicated since the under-constrainedness can be a user *desiderata*, for instance in the case of a specification up to a direct isometry which indeed leads to an infinite number of solutions, the exact location of a solution being irrelevant, or in the case of the specification of a kinematic system, like a pair of scissors.

To tackle under-constrained systems, the main piece of work is that of Joan-Arinyo *et al.* [21, 23]. First of all, they suggest that the main problems for solving under-constrained geometric constraint systems (GCS) are three: completion (add constraints in such a way that the new GCS can be solved by geometric constructions), well-constrained completion (add constraints to an under-constrained GCS so that it becomes well-constrained), and optimal well-constrained completion (add constraints so that the new GCS is well-constrained and the set of equations to solve simultaneously is of minimal size). Second, they propose an algorithm to address the first two problems, by incrementally enriching the constraint graph with new constraints. Among the different possible completions, one needs to find the one that will allow a given geometric solver to solve the completed system. To do this, they use the technique of s-tree decomposition.

Prior to the work of Joan-Arinyo *et al.*, Fudos and Hoffmann [24] proposed a method called cluster formation method, which addresses problems 1 and 3. Lee *et al.* [25] classify under-constrained sub-systems into simplified cases and apply classification rules, both aspects being based on the graph of the GCS, in order to deal with under-constrained systems. The work of Trombettoni *et al.* [26] introduces an algorithm based on an analysis of the degrees of freedom to solve under-constrained GCS. Zhang and Gao [27] proposed a method to address the well-constrained completion problem which can then be used to decompose under-constrained systems.

This paper illustrates the interest of considering under-constrained systems not as constraint systems to be fixed, but rather as systems to be solved as is, in association with tools able to browse the solution space, or as intermediary systems in a solving process. It is organized as follows. Section 2 recall some fundamental definitions and facts of geometric constraint solving. In particular its "original sin" which lies in the invariance under the action of the isometries, makes under-constrained the majority of constraint systems encountered in CAD. Section 3 describes a way to represent and to handle articulated systems. Section 4 explains how the consideration of under-constrained systems obtained by relaxing some constraints may help in solving well-constrained system. We present here two examples: a decomposition method based on the computation of maximal rigid sub-system (here, rigid means well-constrained modulo

the isometries), and a method so-called quasi-decomposition which mixes formal and numerical resolution.

2 Invariance under a global group

In this section, we formalize invariance under the action of transformation groups and show the interest of the multi-group point of view in the context of geometric constraint solving.

2.1 Geometric constraint systems

We use the formalism of geometric constraint systems used in [28]. We briefly recall here the main notions.

A Geometric Constraint System (GCS) is a tuple $\mathcal{S} = (C, X, A)$ with C the set of constraints, X the set of unknowns and A the set of parameters. Given a valuation ρ of the parameters, a figure of $\mathcal{S}$ is a valuation of the elements of X (*i.e.* a map from X to the considered model, generally the Euclidean plane $\mathbf{E_2}$ or the Euclidean space $\mathbf{E_3}$) such that the interpretation of the constraints of C is valid. The set of all figures of $\mathcal{S}$ according to ρ is denoted by $F_\rho(\mathcal{S})$, simply $F(\mathcal{S})$ when values of parameters are not important or F when no confusion occur. Then, $\mathcal{S}$ is well-constrained is $F(\mathcal{S})$ is finite, under-constrained if $F(\mathcal{S})$ is infinite.

The joint operation is the semantical counterpart of system decomposition. Under some compatibility conditions, two figures can be joined. The joint of f_1, defined on X_1, and f_2, defined on X_2, is the figure $f_1 \otimes f_2$ which maps x to $f_1(x)$ if $x \in X_1$ or to $f_2(x)$ if $x \in X_2$. The compatibility conditions are that for any $x \in X_1 \cap X_2$, $f_1(x) = f_2(x)$. The joint operation can be extended to joint of figure sets by considering that $F_1 \otimes F_2$ is the set of all figures obtained by the joint of two compatibles figures f_1 and f_2, respectively in F_1 and F_2. Fig 2 shows the joint of two figure sets (the dotted line expresses symmetry in the top triangle).

The notion of boundary system plays a large part in decomposition of subsystems. A boundary system of a system $\mathcal{S}_1$, subsystem of $\mathcal{S} = \mathcal{S}_1 + \mathcal{S}_2$, contains all the information needed to retrieve the solutions of $\mathcal{S}_2$ which are subfigures of $F(\mathcal{S})$. We denote by $F(\mathcal{S})|_{X_2}$ the subfigures of $F(\mathcal{S})$ restricted to the unknowns set X_2.

The definition of boundary systems is semantical: let $\mathcal{S} = \mathcal{S}_1 + \mathcal{S}_2$ be a system with $\mathcal{S}_1 = (C_1, X_1, A_1)$ and $\mathcal{S}_2 = (C_2, X_2, A_2)$. A boundary system of $\mathcal{S}_1$ with respect to system $\mathcal{S}_2$ is a system that we note $\mathcal{B}_{\mathcal{S}_2}(\mathcal{S}_1) = (C_e, X_e, A_e)$ with $X_e = X_1 \cap X_2$, $A_e = A_1 \cap A_2$ and C_e such that $F(\mathcal{B}_{\mathcal{S}_2}(\mathcal{S}_1)) = F(\mathcal{S}_1)|_{(X_1 \cap X_2)}$. Usually, C_e is computed by heuristics within a specific geometric universe. Again, to clarify notations, when there is no ambiguity as to the system with respect to which the boundary system is computed, it is denoted simply by $\mathcal{B}(\mathcal{S})$. Notice that there may be several different boundary systems, but they are all equivalent to each other.

It can be shown [28, result 2.3] that removing a subsystem $\mathcal{S}_1$ from system $\mathcal{S}$ does not change the solutions of the remaining system if a boundary system of $\mathcal{S}_1$ is added. In other terms, it proves the validity of bottom-up decomposition methods: if the subsystem solvers are correct (*i.e.* yield only figures that satisfy the constraints) then the joint of the subfigures will yield valid solutions.

Let us illustrate this result on the example of Fig. 2: X_2 is the set $\{P_3, P_4, P_5\}$ and $F(\mathcal{S}_2)$ contains all triangles whose two segments are of the same length and angle between them is fixed to parameter a. We can see that $F(\mathcal{S})|_{X_2}$ is the subset of $F(\mathcal{S}_2)$ where distance P_3P_5 of triangles is k_1. $F(\mathcal{S}_2)$ carries triangles that are not involved in any solutions.

The set X_1 is $\{P_1, P_2, P_3, P_5\}$. Boundary variables are $X_e = X_1 \cap X_2 = \{P_3, P_5\}$ and $F(\mathcal{B}(\mathcal{S}_1))$ contains all segments in Euclidean plane where length is k_1. Considering a classical signature, this set can be syntactically expressed by the system $\mathcal{B}(\mathcal{S}_1) = (\{dist\_pp(P_3, P_5, k_1)\}, \{P_3, P_5\}, \{k_1\})$.

Thus, $\mathcal{S}_2 + \mathcal{B}(\mathcal{S}_1)$ restricts $\mathcal{S}_2$ to triangles where the distance of the segment opposite angle $\widehat{P_3P_4P_5}$ is k_1. Notice that $F(\mathcal{B}(\mathcal{S}_2))$ is just all possible segments since boundary variables of $\mathcal{S}_2$ are $\{P_3, P_5\}$ and the distance between these two points is not set. Here, the relation is $F(\mathcal{S})|_{X_1} = F(\mathcal{B}(\mathcal{S}_2)+\mathcal{S}_1)$ but the boundary system $\mathcal{B}(\mathcal{S}_2)$ does not bring relevant informations since $F(\mathcal{S})|_{X_1} = F(\mathcal{S}_1)$. That means that removing $\mathcal{S}_2$ from $\mathcal{S}$ does not impact on X_1.

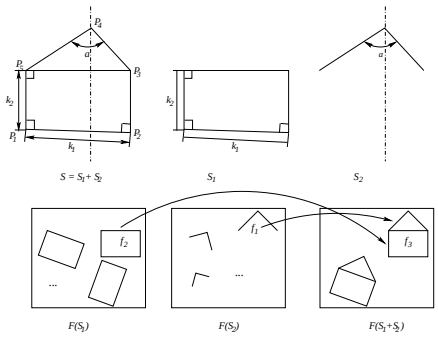


Fig. 2. Joint of two subfigures

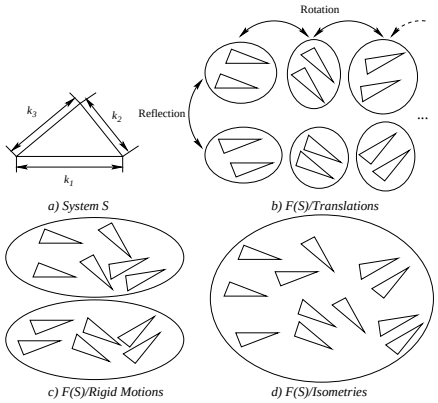


Fig. 3. Orbits of system $\mathcal{S}$ considering different transformations groups

2.2 Transformation groups

A set of figures F is invariant by a group of transformations G (or G-invariant) if for any figure $f \in F$ and any transformation $\varphi \in G$, $\varphi(f) \in F$. By extension, a constraint system $\mathcal{S}$ is said to be G-invariant if $F(\mathcal{S})$ is so. For a group G

and a figure $f \in F$, the set $G.f = \{f' \mid \exists \varphi \in G, \varphi(f) = f'\}$ is the *orbit* of f. The set of orbits of F under the action of G form a partition of F. The associated equivalence relation states that f and f' are equivalent if there exists a transformation $\varphi \in G$ such that $f = \varphi(f')$. The orbits are the equivalence classes of this relation. The set of all orbits of F under the action of G is written as F/G and $|F/G|$ denotes the number of orbits.

Fig. 3a shows the very simple constraint system of a triangle where length of all three sides are given. Given values for parameters k_1, k_2 and k_3, Fig. 3b, 3c and 3d represent the solution set $F(\mathcal{S})$ with different equivalence classes according to the transformations group considered. In Fig. 3b transformations are translations, thus number of orbits is infinite. In Fig. 3c transformations are direct isometries or rigid motions, so $|F(\mathcal{S})/G| = 2$. In the last case, Fig. 3d, there is only one orbit, all solutions are equivalent *modulo* isometries.

When a constraint system is G-invariant, $F(\mathcal{S})$ can be characterized by a set of orbit representatives F_r containing one figure per orbit. In other words, $F(\mathcal{S}) = G.F_r$. In the previous example, the set of solutions can be defined by $F(\mathcal{S}) = G.F_r$ with G the group of rigid motions and F_r a set containing two figures, one from each orbit of $F(\mathcal{S})/G$.

Those definitions allows to define *modulo constrainedness*, according to the number of orbits of the system for a given group: a geometric constraint system $\mathcal{S}$ is said

- under-constrained *modulo* G if $|F(\mathcal{S})/G|$ is infinite,
- well-constrained *modulo* G (or G-well-constrained, G-wc in short) if $|F(\mathcal{S})/G|$ is positive and finite,
- over-constrained if $\mathcal{S}$ has no solutions.

Thus, when G is infinite, a G-well-constrained system is almost always under-constrained since it has an infinite number of solutions in each orbit. Yet, since the position (or the scale, or the orientation) are often irrelevant for the designer, one can call this phenomenon under-constrainedness by abstraction.

A key notion when it comes to *modulo* constrainedness is that of *reference*. A reference for a system is a set of entities (or coordinates of entities) such that pinning down each element of the reference leaves only a finite number of solutions. For instance, a reference for a rigid triangle defined by the lengths of its distances could be a point and a direction from this point to another point. Actually, a point and a direction are a reference for any rigid system and one can similarly define possible reference types for every global group [28]. In the case of an articulated system, a reference anchors each rigid subsystem.

Given a transformation group G and a G-well-constrained system $\mathcal{S}$, let us consider a decomposition of $\mathcal{S}$ under the form $\mathcal{S} = \mathcal{S}_1 + \mathcal{S}_2$. $\mathcal{S}_1$ and $\mathcal{S}_2$ are both G-well-constrained if and only if they share a reference. In other words, in the usual cases either $\mathcal{S}_1$ or $\mathcal{S}_2$ is G-under-constrained and the use of a boundary system, which has to be G-well-constrained, is mandatory.

As said above, the joint operation is very useful for decomposition methods. Considering transformation groups, the definition of the joint operation can be

extended. Two figures f_1 and f_2 can be G-joined if there exist two transformations φ_1 and φ_2 in G such that $\varphi_1(f_1)$ and $\varphi_2(f_2)$ can be joined. The G-joint of f_1 and f_2, noted $f_1 \otimes_G f_2$, is the set of all $\varphi_1(f_1) \otimes \varphi_2(f_2)$.

Notice that a system can be invariant under the action of several groups. For instance, if a system is invariant by displacement, it is also invariant by translation and by rotation. More generally, one can consider a bounded poset of groups and then show that invariance under the action of a given group implies invariance under the action of every included group. The biggest invariance group (in terms of inclusion) of a system is its well-constrainedness group.

2.3 Interest of *modulo* constrainedness

There always exists a group G such that $\mathcal{S}$ is G-wc: the group of permutations of the solution set. Of course, expressing this group means knowing all solutions of $\mathcal{S}$, so this is not useful to solve or decompose. The interesting knowledge is that of the well-constrainedness of a system (or of subsystems) *modulo* global groups, *i.e.* groups of transformation applying the same transformation on each entity of the (sub)system.

Classical solvers consider only GCS which are invariant by displacements, *i.e.* applying a rigid motion to a solution yields another solution. We believe that this assumption weakens geometric constraint solvers.

Indeed, considering subsystem which are invariant under other transformation groups leads to more powerful decomposition algorithms. Schramm and Schreck [29] solve subsystems which are invariant under the actions of similarities (rigid motions + scalings). From this possibiliy, Schreck and Mathis [30] deduce a decomposition algorithm which is able to solve geometric constraint systems which were not considered as decomposable before. Similarly, van der Meiden and Bronsvoort [31] extend classical cluster rewriting approaches by considering, on top of rigid clusters, two types of non-rigid clusters: scalable clusters (*i.e.* sub-systems well-constrained *modulo* similarity, as in [30]) and radial clusters (*i.e.* assemblies of sub-systems which are well-constrained *modulo* similarities but form a cluster which is invariant under the action of similarity but not well-constrained *modulo* similarities).

Moreover, it is important to realize that the user's intent is not always to design a displacement-invariant object. On one hand, the object may be articulated. Articles describing methods allowing to solve articulated GCS are not many and admit the weakness of their resolution power, whether they proceed by analysis of the degrees of freedom of rigid object assemblies [32, 33] or by decomposition into rigid sub-systems [34]. On the second hand, the object may also not be invariant by some rotations. For instance, sketches drawn in the context of architectural design represent objects which are invariant by translation but cannot rotate around x or y axes. Notice that very often, these systems or subsystems can be scaled, when the architect knows the relative distances but does not yet provide a fixed measure.

3 Articulated systems

In this subsection, we propose a method to parameterize an articulated GCS, *i.e.* to find a reference for such a system. We do not address the completion problems exposed in [21] since we do not add constraints to the system and merely explicit which geometric entities should be anchored for the system to have a finite number of solutions. By proceeding this way, we get a homogeneous parameterization method, which can handle a rigid system as well as an articulated system: in the first case, our method will select a point and a direction to be pinned down, for instance.

In this subsection, we consider geometric constraint systems which are under-constrained *modulo* any global transformation group, *i.e.* it is not possible to yield a transformation group which acts likewise on the whole system and for which the number of orbits is finite. Eventhough they are under-constrained, many such systems define final objects as they are intended by the user. This is the case of all articulated objects: a pair of scissors, a desklamp, a car or any robotic system.

In order to solve these systems, three steps are needed:

1. find a parameterization of the GCS, *i.e.* a reference,
2. determine the possible values of the parameters,
3. draw the solutions for given values of the parameters.

We focus here on step 1 and use work described in the literature [17, 18] for the other two steps. We present an incremental algorithm to compute a parameterization of a GCS $\mathcal{S} = (C, X, A)$. The idea of this algorithm is to consume constraints one by one and consider a generic reference for the system induced by the new constraint. This generic reference is modified according to what the system shares with the system generated by the previously consumed constraints.

3.1 Parameterization algorithm

The reference is computed under the form of a directed acyclic graph (DAG). The orientation of the DAG indicates which parts of the order in which the different parts of the reference must be anchored. When there are several disconnected subsytems, we consider several DAGs. Together with the DAGs come the construction rules which indicate how to build entities of the system with parts of the reference as arguments.

Algorith 1 gives the general process of the algorithm. At each iteration, a new constraint c is taken into account, which will be added to the already constructed system $\mathcal{S}'$, initially empty. During the addition of c, we start by computing the system induced by c, $\mathcal{S}_c$ and the boundary system $\mathcal{B}$ of $\mathcal{S}_c$ with regard to $\mathcal{S}'$. Several cases occur:

1. $\mathcal{B}$ is empty,
2. $\mathcal{B}$ contains only geometric entities

3. $\mathcal{B}$ contains constraints concerning geometric entities from disconnected subsystems

4. $\mathcal{B}$ contains constraints concerning several geometric entities from a subsystem

Algorithm 1 Incremental parameterization algorithm

input: $\mathcal{S} = (C, X, A)$, a geometric constraint system
output: R, a reference for $\mathcal{S}$ (DAG)

```
1: let S' = be an empty GCS and R an empty DAG
2: for all c ∈ C do
3:     let S_c be the GCS induced by c and r_c its reference
4:     let B = (C_B, X_B, A_B) be the boundary of S_c with regard to S'
5:     if B is empty then
6:         R ← R + r_c //case 1
7:     else if C_B is empty then
8:         R ← R + (r_c − X_B) //case 2
9:     else
10:        if no two entities of X_B belong to a same connected component of the
           constraint graph of S' then
11:            remove the reference of all concerned connected components except one
               //case 3
12:            add part of r_c as in case 2
13:            add each previously removed reference as in case 2
14:        else
15:            compute the boundary system of the connected component containing
               several entities concerned by c //case 4
16:            if it contains the same constraint then
17:                if the metric is the same, then the constraint is redundant, else it
                   overconstrains the system endif
18:            else
19:                attempt to detect rigidification and to correct R using geometric
                   construction rules
20:            end if
21:        end if
22:    end if
23:    if c was detected as over-constraining or redundant then
24:        output an error/warning message
25:    else
26:        S' ← S' + S_c
27:    end if
28: end for
```

Case 1 : empty boundary

If the boundary is empty, it means the geometric entities concerned by the new constraint do not yet appear in $\mathcal{S}'$: a new connected component is created by adding $\mathcal{S}_c$ and r_c can be added as is.

Case 2 : only entities in the boundary

If the boundary contains only entities, it means the new constraint only partially concerns entities of $\mathcal{S}'$ and the already computed reference R does not need modifications. r_c is computed and we remove from it entities which are in the boundary, already given by $\mathcal{S}'$. For instance, if the new constraint is a point-point distance, r_c consists in one point and one/two directions, according to the dimension of the geometric universe. If one of these points is already in $\mathcal{S}'$, then only the directions are added to R.

Case 3 : the boundary contains a constraint between disconnected subsystems

If $\mathcal{B}$ contains the added constraint but the concerned entities are not in the same connected component of the constraint graph, it means that the new constraints links two independent subsystems $\mathcal{S}_1$ and $\mathcal{S}_2$ (or more in case of non-binary constraints but for the sake of clarity, we here consider there are only two subsystems). In this cas, we first add $\mathcal{S}_c$ to one of these systems, say $\mathcal{S}_1$ (the choice is made through a heuristic, ours being to choose the system with the biggest reference), thus computing the addition to the reference as in case 2.

Then, we "reverse" the reference DAG of $\mathcal{S}_2$ so that its base parameter is the entity concerned by c. Connecting this DAG is then easy as one simply removes the base parameter to make both DAG compatible.

Reversing the DAG is performed by considering a temporary DAG, initially empty, and by "adding" the boundaries of each rigid subsystem in the right order.

Case 4 : the boundary contains a constraint between two entities of a same subsystem

This case occurs either when a closed chain is created or when a redundant constraint is added. The latter is detected by computing the boundary of $\mathcal{S}'$ with regard to $\mathcal{S}_c$. If it also contains c we detect a redundancy (if the metrics are consistent) or an over-constrainedness.

If the constraint is not redundant, we consider classical construction rules to correct the DAG. These classical construction rules are, for instance "if a point is concerned by two/three distance constraints in 2D/3D, it can be constructed by intersection of the corresponding circles/spheres". These rules allow us to remove elements from the reference DAG when needed. For instance, consider a three-bars articulated object and the addition of a constraint closing it. One of the extremities of the three-bars object will be considered as built using a

construction rule, thus removing the part of the reference that previously allowed its construction. The new DAG contains a point and two directions, so as to build two of the bars. The rest of the system depends entirely on these bars.

The rigidification of a subsystem can be detected: when the parameters of a construction rule are all in the same rigid subsystem, the constructed elements extend this rigid subsystem.

It may happen that no construction rule can be found. For instance, consider one of the $K_{3,3}$ systems of figure 4: until the last constraint is added, our algorithm works fine. The last constraint rigidifies the system but we cannot detect it since no geometric rule applies. In this particular case, we consider the corresponding subsystem as rigid eventhough we cannot yield a construction plan and use a reparameterization method (see section 4.3) for which we already know the constraint to remove.

3.2 Limits of this algorithm

An important limit of this algorithm lies in the fact that some rigid systems are only identified through the heuristic consisting in considering that a system with 4 degrees of freedom becomes rigid if we add a new non redundant constraint. Mathematical theorems could trick the algorithm.

Algorithm 1 yields only one reference for a given system. This specific reference may not be the one the user wants and we do not yet have a way to provide a reference with several mandatory elements in it: only the base parameter can be chosen, by using the reversal method. Even then, the cost of this reversal is heavy, since it consists in adding the boundaries of each rigid subsystem, thus restarting the construction of non-rigid closed chains from scratch. Also, this algorithm considers assemblies of rigid subsystems, and cannot take into account subsystems which are well-constrained *modulo* other groups than the rigid motions. A promising track for these drawbacks lies in flow-based methods [35], but there is a risk that their use only makes worse the problems of the combinatorial heuristic mentioned above, especially for the detection of over-constrainedness. Another track is a parallel use of the witness method [36–38].

4 Decomposition and under-constrainedness

Decomposing constraint systems into sub-system is an avatar of the "divide and conquer" paradigm which leads to several methods whose goal is to make the geometric solvers more powerful. For instance, we can cite cylindrical decomposition, Gröbner bases computation, König-Hall decomposition, maximal flow based methods, decomposition into triconnected components, geometric knowledge based systems, clusters, *etc.*

As mentioned above, under-constrainedness is often a consequence of the decomposition process when its deal with invariance under the action of a global group (consider, for instance, the notion of virtual bond used into the Owen decomposition). We will see in the next sections, two example of methods where under-constrainedness is part of the decomposition process itself.

4.1 Decomposition and solving

Let us recall more precisely what constraint systems decomposition means: given a constraint system $\mathcal{S}$, search a system $\mathcal{S}'$ such that :

- $\mathcal{S}'$ is semantically equivalent to $\mathcal{S}$,
- $\mathcal{S}'$ is the union of two or more simpler systems
  - which are well-constrained modulo some global group,
  - and not reduced to their boundary,
- solving $\mathcal{S}'$ by using a joint operation is easier, and yields the solutions for $\mathcal{S}$

The more powerful decomposition methods come from the algebraic elimination theory [39]. Indeed given an algebraic system, they are able to yield one or several algebraic systems under triangular form. After that, each equation has to be solved using algebraic method, like Lebesgue's method, or numerical method. But the exponential complexity of algebraic formal method disqualify them for problems of practical use.

This is why, in CAD, combinatorial methods are considered. We give here the example of the so-called propagation of degrees of freedom (DOF propagation) which is perhaps the more basic decomposition method. Starting from a reference, the method tries to iteratively add to the system $\mathcal{S}_1$ an unknown x and constraints defining x until $\mathcal{S}_1$ is equal to $\mathcal{S}$. More precisely, this method translates a constraint system $\mathcal{S}$ into a *constraint hyper-graph* $H = (V, E)$ where the vertexes correspond to the unknowns labeled by their degree of freedom, roughly speaking their number of coordinates, and the hyper-edges correspond to the constraints labeled by their degree of restriction roughly speaking the number of real equations subtended by the constraint. The forward chaining version of the method can be summarized by algorithm 2. At the end of this process, $\mathcal{S}_1$ corresponds to a subsystem which is structurally well-constrained, and `L` describes a way to decompose $\mathcal{S}_1$. The algorithm is said successful when $\mathcal{S} = \mathcal{S}_1$. This method yields a simple planing method where the unknowns are tacking into account one after the other.

This DOF propagation method is based on a local application of the Laman principle and find a well-constrained subsystem $\mathcal{S}'$ of $\mathcal{S}$. This subsystem is *maximal according to this method*. Unfortunately, DOF propagation is not powerful, even in 2D. Indeed, sometimes $\mathcal{S}'$ is reduced to a single constraint regardless the chosen reference and the method fails to perform an interesting decomposition (see, for instance, the $K_{3,3}$ problem above). Moreover, like most of the combinatorial solvers, this method is tricked by dependences induced by geometrical theorems. for instance, it is unable to detect that a triangle constrained by three angle constraints is over-constrained.

The W-decomposition method is also based on the computation of a maximal subsystem well-constrained modulo the displacements. Based on the witness notion (see ...), it is not tricked by unwanted dependences, and it is much more powerful than combinatorial methods in computing well-constrained subsystems.

Algorithm 2 Simple DOF propagation int he case of all geometric entities have DOF 2, and each edge represents a constraint with DOC 1

input: G, the constraint (hyper)-)graph corresponding to $\mathcal{S}$
output: L an ordered list of vertexes of G
1: L is the list of the visited vertexes,
2: S_1 is the list of the visited edges
3: L = choose vertexes of G corresponding to a reference
4: S_1 = the constraints joining the chosen vertexes
5: **repeat**
6: choose edges $\{e_1, \dots e_m\}$ corresponding to $\{c_1, \dots c_m\}$, such that
- they are not in S_1,
- they have all the same unknow x as extremity,
- they other extremities are in L,
- $\text{dof}(x) = \Sigma_i \text{dor}(c_i)$
7: **if** such edges exist **then**
8: add x to L and,
9: add $\{e_1, \dots e_m\}$ to S_1
10: **end if**
11: **until** no more edges can be chosen

4.2 W-decomposition

The idea of applying the notion of *witness* in GCS to CAD/CAM problems comes from D. Michelucci. It was explained in several places, including the 2006 ADG workshop [37].

Considering a witness allows to find a maximal rigid sub-system (MRS) of a constraint system (see [36, 37]). Applying this method to a rigid constraint system $\mathcal{S}$ yields whole system $\mathcal{S}$, but considering an underconstrained subsystem of $\mathcal{S}$, some rigid parts of $\mathcal{S}$ can be retrieved: the W-decomposition is a decomposition method based on this fact.

The basic idea of W-decomposition is then to remove constraints from $\mathcal{S}$, giving system $\mathcal{S}'$, and see if $\mathcal{S}'$ can be broken into non-trivial MRSs, *i.e.* MRSs which are not limited to their boundary. If it does, then we use W-decomposition on each non-trivial MRS. Algorithm 3 gives the pseudo-code of the algorithm.

Efficiency of the execution depends on the choice of the removed constraint. In the worst case, all constraints are tested: $2 \times n - 3$ uses of the MRS algorithm are made, thus the complexity is $\mathcal{O}(n^4)$.

It must be noticed that W-decomposition does not fail because of the connectivity of the constraint graph: For instance, Fig. 5a gives an example of a 4-connected constraint graph which is W-decomposable, no matter what is inside the inner blue part as long as it is rigid. Moreover, W-decomposition is not based on a bottom-up computation, like methods with clusters, and succeeds in decomposing the systems correponding to the graphs of Fig. 4 and Fig. 5b which are not decomposable by classical combinatorial methods.

W-decomposition is a mixture of combinatorial and numerical random methods. It is obviously not as powerful than algebraic methods, for instance it fails

Algorithm 3 W-decomposition

Input: a rigid GCS $\mathcal{S}$ with
its constraint graph $G = (V, E)$ and
a witness W of $\mathcal{S}$
Output: a list of rigid subsystems
repeat
  Delete a constraint e
  Identify MRSs of $(V, E/\{e\})$ using a witness
  while each MRS is equivalent to its boundary **do**
    Choose another constraint e and identify MRSs of $(V, E/\{e\})$
  end while
until all constraints are tested or there is a MRS which is not equivalent to its boundary
if no MRS bigger than its boundary is found **then**
  return list [G] *//G is W-indecomposable*
else
  remove all the constraints included in non-trivial MRSs
  insert the boundary of all non-trivial MRSs in the system
  reintroduce constraint e in the system*//this gives a rigid constraint system*
  recursively W-decompose the resulting system
  recursively W-decompose all previously identified MRSs
  return the concatenation of the lists obtained in the last two lines
end if

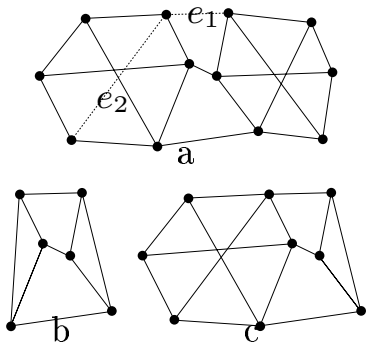


Fig. 4. 2D systems where edges represent point-point distances; a: 3-connected constraint graph made of two $K_{3,3}$ graphs connected with 3 constraints; b and c: graphs obtained by replacing MRSs identified by algorithm 3 by their boundary with respectively edges e_1 and e_2 removed.

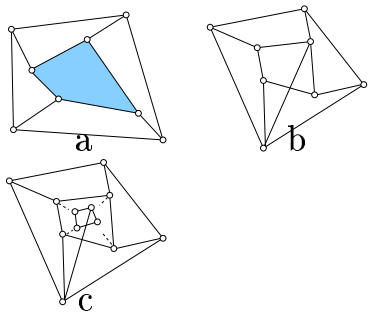


Fig. 5. 2D examples for the W-decomposition: each vertex is a point and each edge represents a distance constraint. a: W-decomposable 4-connected GCS (the blue subsystem is rigid); b: W-indecomposable system; c: there are W-indecomposable systems with an arbitrary number of points.

in decomposing all the systems depicted in Fig. 5c. But, we feel that it owns one of the best ratio decopmposition power/efficiency for constraint systems coming from CAD.

4.3 Quasi-decomposition

We now outline a work about decomposition which was already presented at an ADG workshop and published in [40, 41]. This method use an under-constrained system which is built on the fly and is considered in some sense as an articulated system whose some solutions are browsed through a numerical method. Roughly speaking, the idea beyond the first step of the method consists in examining why the decomposition method fails, and, in relaxing some constraints to make it succeed.

It is useful to come back to the DOF propagation and to notice that the failure of the method does not necessarily implies that system $\mathcal{S}$ is miss-constrained. More precisely, assuming that system $\mathcal{S}_1$ is well-constrained, the case $\text{dof}(x) < \Sigma_i \text{dor}(c_i)$ (which makes fail the algorithm 2 at line 6, fourth item) implies that the whole system $\mathcal{S}$ is over-constrained: in other words, there is a bug in the designer specification. But the method could as well fail, even if system $\mathcal{S}$ is well-constrained, if for all unknowns x not in $\mathcal{S}_1$, $\text{dof}(x) > \Sigma_i \text{dor}(c_i)$: consider for instance the $K_{3,3}$ graph given at Fig. 4.

Then, recall that decomposition of system $\mathcal{S}$ implies to compute a system $\mathcal{S}'$ semantically equivalent to $\mathcal{S}$ and under the form of the union of smaller subsystems. Relaxing the equivalence condition between $\mathcal{S}$ and $\mathcal{S}'$ leads to a more flexible scheme which we call quasi-decomposition: given a constraint system $\mathcal{S}$, search for a system $\mathcal{S}'$ such that :

- $\mathcal{S}$ and $\mathcal{S}'$ do not necessarily have the same solutions but are “similar" in some sense
- $\mathcal{S}'$ is decomposable,
- there is a way to transform any solution for $\mathcal{S}'$ into a solution for $\mathcal{S}$ and one hopes that any solution for $\mathcal{S}$ can be obtained like that.

This scheme was used by the adaptation of the homotopy method to CAD problems (see [42]). But it was also used, in a different way, by Gao *et al.* in the case where $\mathcal{S}$ is “quasi" decomposable with respect to the DOF propagation method.

Exploiting the decomposition failures to modify a system Whatever the decomposition method used, when an irreducible constraint system $\mathcal{S}$ is considered, some room for manœuvre can be obtained by forgetting some constraints of $\mathcal{S}$, what gives an under-constrained system, say $\mathcal{S}_1$. Making $\mathcal{S}_1$ well-constrained while keeping flexibility can be done in two ways: transforming some unknowns into parameters [43], or adding constraints with parameters [44]. Obviously, the latter way is more general since transforming unknown x into parameter μ results in adding equation $x = \mu$. In this case, parameterized constraints are added

to the system, and we can make their parameters vary in compensation of the missing constraints.

More formally, we have the following scheme (for the sake of simplicity, we consider the replacement of a single constraint):

$$\mathcal{S} = (\begin{cases} c_1(x_1, \ldots, x_p) \\ \ldots \\ c_{m-1}(x_1 \ldots, x_p) \\ c_m(x_1 \ldots, x_p) \end{cases} ; \{x_1, \ldots, x_p\}; \emptyset)$$

gives system $\mathcal{S}_1$ by forgetting, for instance, c_m:

$$\mathcal{S}_1 = (\begin{cases} c_1(x_1, \ldots, x_p) \\ \ldots \\ c_{m-1}(x_1 \ldots, x_p) \end{cases} ; \{x_1, \ldots, x_p\}; \emptyset)$$

which gives by adding constraint d with parameter k:

$$\mathcal{S}' = (\begin{cases} c_1(x_1, \ldots, x_p) \\ \ldots \\ c_{m-1}(x_1 \ldots, x_p) \\ d(x_1, \ldots x_p; k) \end{cases} ; \{x_1, \ldots, x_p\}; \{k\})$$

or, in short, $\mathcal{S}' = \mathcal{S} - c_m(x_1, \ldots x_p + d(x_1, \ldots x_p; k)$. The addition of one or more parameters imposes $\mathcal{S}'$ to be formally solved and the question arises of choosing the constraints to be forgotten such that $\mathcal{S}'$ is formally solvable.

The answer of this question is strongly related to the nature of the formal solver used. For instance, in [44], the authors assume that their formal solver can solve any geometric construction problem with one unknown object and then, they use a systematic search of the constraints to be eliminated. This ensures that the remaining system is solvable and a minimal number of constraints is discarded. But this brute force algorithm is not usable anymore when there are more than two constraints to remove and when the solver is not able to formally solve any construction problem with one unknown.

The backward chaining consists in choosing the less constrained object of the system, call it x_{i_0}: an heuristic choice is made in the case where there are several such objects. We call the neighborhood of x_{i_0} the set of all constraints involving x_{i_0} and we note it $N(x_{i_0})$. Assuming that all the other objects are known, the formal solver is employed to construct x_{i_0} by using constraints in $N(x_{i_0})$:

- if the solver fails to determine x_{i_0} then
  - try another few constrained object,
  - if the solver fails in any case, then try to add some heuristically chosen parametric constraints d_j;
- if the solver succeeds to formally determine x_{i_0} from its neighbors, two cases can occur:
  - if all constraints of $N(x_{i_0})$ are used, the chaining process can continue
  - if some constraints of $N(x_{i_0})$, say c'_l, are not used, then keep them apart.

Algorithm 4 The algorithm applying one step of a "one step" transactional expert system

```
input : B, base of unused constraints
    Se, a transactional expert system
    S1, the constraint system to be analyzed
output : ok, a boolean
    B, the modified base of constraints
    S1, the modified constraint system
 1: l = extract_obj_dof(B) // unknowns list
 2: ok = false
 3: (continue,o,r,B+,B-) = try(B, Se, S1)
 4: // try to apply a rule removing r degrees of
 5: // freedom from o, adding facts in B+ and
 6: // removing facts in B- (B is unchanged)
 7: while continue do
 8:     x = current_DOF(o,l)
 9:     if x+r > dof(o) then
10:         continue = false
11:         store all the supernumerary constraints
12:     else
13:         if x+r = dof(o) then
14:             B = update(B,B+,B-)
15:             continue = false, ok = true
16:             S1 = update_sys(S1, B)
17:         else
18:             l = update_list(l, (o,x+r))
19:             B = update(B,B+,B-)
                // try another rule/object
20:             (continue,o,r,B+,B-) = try(B,Se,S1)
21:         end if
22:     end if
23: end while
24: done
25: return (ok, B, S1)
```

The system $\mathcal{S}'$ to be considered is $\mathcal{S} - \sum_l c'_l + \sum_j d_j$ (see Alg. 4)

Conversely, the forward chaining strategy consists in letting the knowledge based solver act on the constraint system. When the solver fails to determine one unknown because some constraints are missing, then it tries to add some parametric constraints d_j in order to continue. At the end, when all the unknowns are solved, there are some constraints c'_l which have not been used since system $\mathcal{S}$ was well-constrained: these constraints must be forgotten. Once again, the system $\mathcal{S}'$ to be considered is $\mathcal{S} - \sum_l c'_l + \sum_j d_j$.

This way, the resolvability of $\mathcal{S}'$ by the formal solver is ensured. In principle, as many constraints were removed as ones were added (more precisely, we should consider the degree of restriction of the constraints): we call *syntactic distance* between $\mathcal{S}$ and $\mathcal{S}'$ the number of removed constraints. The heuristics used in the previous strategies aim at minimizing this distance, that is, at modifying $\mathcal{S}$ as little as possible.

Solving Let d be the syntactic distance between $\mathcal{S}$ and $\mathcal{S}'$. $\mathcal{S}'$ is a parametric constraint system with d parameters $k_1, \dots k_d$ and p unknowns $x_1, \dots x_p$. Since $\mathcal{S}'$ is formally solvable in our framework, we can use its solutions which are functions $f_i : (k_1, \dots k_d) \mapsto x_i(k_1, \dots k_d)$ where $i = 1, \dots p$.

In addition, $\mathcal{S}$ and $\mathcal{S}'$ differ by d constraints. More precisely, most solutions of $\mathcal{S}'$ do not satisfy the removed constraints $c'_1, \dots c'_d$. But, we can search for values for the parameters k_i which satisfy these constraints. In fact, we just have to solve the system $\mathcal{S}_2$:

$$\begin{cases} c'_1(f_1(k_1, \dots k_d), \dots, f_p(k_1, \dots k_d)) \\ \dots \\ c'_d(f_1(k_1, \dots k_d), \dots, f_p(k_1, \dots k_d)) \end{cases}, \{k_1, \dots k_d\}, \emptyset$$

whose unknowns are $\{k_1, \dots k_d\}$ and without parameters. Thus, $\mathcal{S}_2$ can be solved numerically. It is then clear that if $(v_1, \dots v_d)$ is a solution for $\mathcal{S}_2$, then

$$(f_1(v_1, \dots v_d), \dots f_p(v_1, \dots v_d))$$

is a solution for $\mathcal{S}$. On certain conditions about the added constraints d_j, it can be proved that if $\mathcal{S}_1$ is well-constrained, all the solutions can be obtained this way if both solvers are complete.

Some difficulties arise when a numerical solver is used to solve system $\mathcal{S}_2$ but it is beyond the scope of this paper. Interested readers may refer to [41].

5 Conclusion

We explained in this paper why under-constrained systems are useful in CAD: they naturally appear when constraint systems are invariant *modulo* an infinite group of transformations, but they also are an ingredient of some decomposition methods.

Moreover, from the final user point of view, it is quite frustrating, when there are an infinite number of solutions, not to see any of them. This occurs when the user wants to specify an articulated object or when in an incremental process of design, the user wants to see the evolution of the solutions. This is why, we think that solvers should be able to deal with under-constrained systems by proposing some particular solutions for the current system, by giving tools for browsing infinite space of solutions. These features could make way for a new generation of CAD softwares, more easily accessible to non-expert users. We are currently working on a prototype taking these points into account.

References

1. Laborde, C.: Integration of technology in the design of geometry tasks with Cabri-geometry. International Journal of Computers for Mathematical Learning **6**(3) (2002) 283–317
2. Kortenkamp, U., Richter-Gebert, J.: The Interactive Geometry Software Cinderella.2. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2007)
3. Heyting, A.: Axioms for intuitionistic plane affine geometry. In L. Henkin, P.S., Tarski, A., eds.: The axiomatic Method, with special reference to Geometry and Physics, Amsterdam, North-Holland (1959) 160–173
4. Dehlinger, C., Dufourd, J.F., Schreck, P.: Higher-order intuitionistic formalization and proofs in Hilbert's elementary geometry. In: 3rd International Workshop on Automated Deduction in Geometry, Springer, LNAI 2061 (2000) 306–323
5. Narboux, J.: A decision procedure for geometry in Coq. In: TPHOL'04: Proceedings of the 2006 conference on Theorem Proving and Higher Order Logic. Volume 3223 of LNCS., Springer-Verlag (2004) 225–240
6. Bezem, M., Hendriks, D.: On the Mechanization of the Proof of Hessenberg's Theorem in Coherent Logic. Journal of Automated Reasoning **40**(1) (2008) 61–85
7. Schreck, P.: Automatisation des constructions géomètriques à la règle et au compas. PhD thesis, Université Louis Pasteur, Strasbourg 1 (January 1993)
8. Schreck, P.: Robustness in CAD geometric construction. In: IV'01: Proceedings of the fifth International Conference on Information Visualisation, London, UK, IEEE Computer Society (2001) 111–116
9. Aldefeld, B.: Variations of geometries based on a geometric-reasoning method. Computer-Aided Design **20**(3) (1988) 117–126
10. Dufourd, J.F., Mathis, P., Schreck, P.: Geometric construction by assembling solved subfigures. Artificial Intelligence **99**(1) (1998) 73–119
11. Ait-Aoudia, S., Jegou, R., Michelucci, D.: Reduction of constraint systems. In: Proceedings of the Compugraphics Conference. (1993) 83–92
12. Bouma, W., Fudos, I., Hoffmann, C., Cai, J., Paige, R.: A geometric constraint solver. Computer-Aided Design **27**(6) (1995) 487–501
13. Owen, J.: Algebraic solution for geometry from dimensional constraints. In: SMA'91: Proceedings of the 1st ACM Symposium of Solid Modeling and CAD/CAM Applications, ACM Press (1991) 397–407
14. Hoffmann, C., Lomonosov, A., Sitharam, M.: Geometric constraint decomposition. In Brüderlin, B., Roller, D., eds.: Geometric Constraint Solving and Applications. Springer (1998) 170–195

15. Jermann, C., Trombettoni, G., Neveu, B., Rueher, M.: A constraint programming approach for solving rigid geometric systems. In: Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, Springer-Verlag (2000) 233–248
16. Jermann, C., Trombettoni, G., Neveu, B., Mathis, P.: Decomposition of geometric constraint systems: a survey. IJCGA **16**(5,6) (2006) 379–414
17. Luzón, M.V., Soto-Riera, A., Gálvez, J.F., Joan-Arinyo, R.: Searching the solution space in constructive geometric constraint solving with genetic algorithms. Applied Intelligence **22**(2) (2005) 109–124
18. Sitharam, M., Arbree, A., Zhou, Y., Kohareswaran, N.: Solution space navigation for geometric constraint systems. ACM Transactions on Graphics **25**(2) (2006) 194–213
19. Villard, C., Schreck, P., Dufourd, J.F.: Sketch-based pruning of a solution space within a formal geometric constraint solver. Artificial Intelligence Journal **124**(1) (2000) 139–159
20. van der Meiden, H., Broonsvort, W.: An efficient method to determine the intended solution for a system of geometric constraints. International Journal of Computational Geometry and Applications **15**(3) (2005) 279–298
21. Joan-Arinyo, R., Soto-Riera, A., Vila-Marta, S., Vilaplana-Pasta, J.: Transforming an under-constrained geometric constraint problem into a well-constrained one. In: SMA'03: Proceedings of the eighth ACM symposium on Solid modeling and applications, New York, NY, USA, ACM Press (2003) 33–44
22. Gao, H., Sitharam, M.: Combinatorial classification of 2D underconstrained systems (2005) Available on the web at the URL: http://www.cise.ufl.edu/ sitharam/under.pdf.
23. Joan-Arinyo, R., Soto-Riera, A., Vila-Marta, S., Vilaplana-Pasto, J.: Revisiting decomposition analysis of geometric constraint graphs. Computer-Aided Design **36**(2) (2004) 123–140
24. Fudos, I., Hoffmann, C.M.: A graph-constructive approach to solving systems of geometric constraints. ACM Transactions on Graphics **16**(2) (1997) 179–216
25. Lee, K.Y., Kwon, O.H., Lee, J.Y., Kim, T.W.: A hybrid approach to geometric constraint solving with graph analysis and reduction. Advances in Engineering Software **34**(2) (2003) 103–113
26. Trombettoni, G., Wilczkowiak, M.: GPDOF: a fast algorithm to decompose underconstrained geometric constraints: Application to 3D modeling. International Journal of Computational Geometry and Applications **16**(5-6) (2006) 479–511
27. Zhang, G.F., Gao, X.S.: Well-constrained completion for under-constrained problems. International Journal of Computational Geometry and Applications **16**(5,6) (2006) 461–478
28. Mathis, P., Thierry, S.E.B.: A formalization of geometric constraint systems and their decomposition. Formal Aspects of Computing **Online first** (2010)
29. Schreck, P., Schramm, E.: Using the invariance under the similarity group to solve geometric constraint systems. Computer-Aided Design **38**(5) (2006) 475–484
30. Schreck, P., Mathis, P.: Geometrical constraint system decomposition: a multigroup approach. International Journal of Computational Geometry and Applications **16**(5,6) (2006) 431–442
31. van der Meiden, H.A., Bronsvoort, W.F.: A non-rigid cluster rewriting approach to solve systems of 3D geometric constraints. Computer-Aided Design **42**(1) (2010) 36–49

32. Kramer, G.A.: Using degrees of freedom analysis to solve geometric constraint systems. In: SMA'91: Proceedings of the 1st ACM Symposium of Solid Modeling and CAD/CAM Applications, ACM Press (1991) 371–378
33. Kramer, G.A.: A geometric constraint engine. Artificial Intelligence **58**(1-3) (1992) 327–360
34. Thierry, S., Mathis, P., Schreck, P.: Towards an homogeneous handling of under-constrained and well-constrained systems of geometric constraints. In: SAC'07: Proceedings of the 2007 ACM Symposium on Applied Computing, Seoul, Korea, ACM Press (2007) 773–777
35. Latham, R., Middleditch, A.: Connectivity analysis : a tool for processing geometric constraints. Computer-Aided Design **28**(11) (1996) 917–928
36. Michelucci, M., Foufou, S.: Geometric constraint solving: The witness configuration method. Computer-Aided Design **38**(4) (2006) 284–299
37. Michelucci, M., Foufou, S.: Detecting all dependences in systems of geometric constraints using the witness method. In: ADG '06: Proceedings of the sixth international workshop on Automated Deduction in Geometry. Volume 4869 of Lecture Notes in Artificial Intelligence., Pontavedra, Spain, Springer (2007) 98–112
38. Michelucci, D., Foufou, S.: Interrogating witnesses for geometric constraint solving. In: SMA '09: Proceedings of the SIAM/ACM joint conference on Geometric and Physical Modeling, San Francisco, California, USA, ACM (2009) 343–348
39. Cox, D.A., Little, J., O'Shea, D.: Using algebraic geometry. Volume 185 of Graduate Texts in Mathematics. Springer (1998)
40. Fabre, A., Schreck, P.: Combining symbolic and numerical solvers to simplify indecomposable systems solving. In: SAC'08: Proceedings of the 2008 ACM symposium on Applied computing, Fortaleza, Ceara, Brazil, ACM Press (2008) 1838–1842
41. Fabre, A.: Contraintes géométriques en dimension 3. PhD thesis, Université Louis Pasteur, Strasbourg 1 (november 2006)
42. Lamure, H., Michelucci, D.: Solving geometric constraints by homotopy. IEEE Transactions on Visualization and Computer Graphics **2**(1) (1996) 28–34
43. Gao, X.S., Hoffmann, C., Yang, W.Q.: Solving spatial basic geometric constraint configurations with locus intersection. In: SMA'02: Proceedings of the 7th ACM Symposium on Solid Modelling and Applications, Saarbrücken, ACM Press (2002) 95–104
44. Gao, X.S., Hoffmann, C., Yang, W.Q.: Solving spatial basic geometric constraint configurations with locus intersection. Computer-Aided Design **36**(2) (2004) 111–122