Decomposition of Geometric Constraint Systems: a Survey

Christophe Jermann, Gilles Trombettoni, Bertrand Neveu, Pascal Mathis

International Journal of Computational Geometry & Applications

Decomposition of Geometric Constraint Systems: a Survey

Christophe Jermann[1] Gilles Trombettoni[2] Bertrand Neveu[2] Pascal Mathis[3]

[1] *LINA - Université de Nantes*
2 rue de la Houssinière BP92208, F-44322 Nantes cedex 3, France
Christophe.Jermann@lina.univ-nantes.fr

[2] *COPRIN team, INRIA Sophia Antipolis*
2004 route des Lucioles BP93, F-06902 Sophia Antipolis cedex, France
[neveu, trombe]@sophia.inria.fr

[3] *LSIIT - Université Louis Pasteur*
Pôle API, boulevard Sébastien Brant, F-67400 Illkirsch, France
mathis@dpt-info.u-strasbg.fr

Significant progress has been accomplished during the past decades about geometric constraint solving, in particular thanks to its applications in industrial fields like CAD and robotics. In order to tackle problems of industrial size, many solving methods use, as a preprocessing, decomposition techniques that transform a large geometric constraint system into a set of smaller ones.

In this paper, we propose a survey of the decomposition techniques for geometric constraint problems[a]. We classify them into four categories according to their *modus operandi*, establishing some similarities between methods that are traditionally separated. We summarize the advantages and limitations of the different approaches, and point out key issues for meeting industrial requirements such as generality and reliability.

Keywords: geometric constraints; decomposition techniques; rigidity; DOF analysis; connectivity analysis; DR-planner; maximum matching; PDOF; WCM.

1. Introduction

The decomposition of constraint systems is an avatar of the well-known divide and conquer paradigm. Applied to geometric constraint systems, it gives birth to general algorithms that answer to the following questions:

- How to cut a system into several smaller subsystems easier to solve?
- How to solve every subsystem separately?
- How to merge the solutions of subsystems for producing the solutions to the whole system?

Preliminary example

Let us consider the simple example shown in Fig. 1 made of points A, B, C, D and lines $\delta_1, \delta_2, \delta_3$. This dimensioned sketch (left side) can be cut into two parts (right

[a]The French CNRS has supported the working groups "AS contraintes géometriques" and "Math-STIC geometric solvers".

side): the subsystem (1) and the subsystem (2).

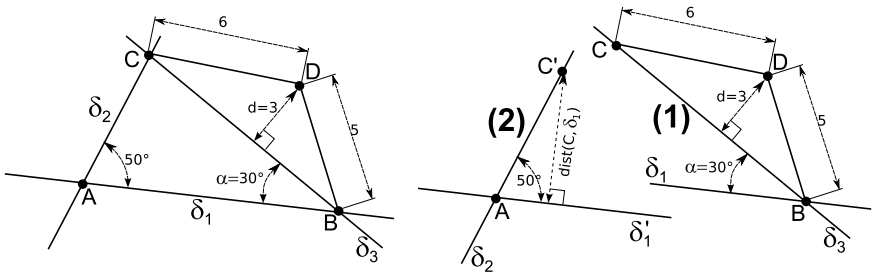

Fig. 1. **Left:** A simple dimensioned sketch. **Right:** A decomposition into two subsystems.

In the literature, several ways to produce such a decomposition have been described. Some algorithms identify subsystem (1) as an interesting (i.e., solvable) subsystem; other methods find *points of weakness* in the constraint system like (C, δ_1); other techniques identify first the subsystem (2) as solvable provided that the complementary subsystem is solvable as well.

This example also shows an interesting phenomenon. The subsystem (2) is not rigid and a metric information coming from the subsystem (1) has to be added, namely the distance between C' and δ_1' (dotted in Fig. 1). This implies an order: the subsystem (1) has to be solved first to determine the value of this distance.

Usually, a geometric subsystem is translated into a system of algebraic equations that is handled by standard numerical or symbolic solving methods. Other solving methods work at the geometric level by applying well-known construction steps, like ruler&compass ones. When subsystems are well-constrained up to displacements, the algorithms need to *pin* some elements (e.g., the point D and the direction of the line δ_3 in the subsystem (1)) in order to make them solvable.

Once the subsystems have been solved, the *subsolutions* are assembled. In our example, this is done using geometric transformations that make coincide point C' with point C, and line δ_1' with line δ_1.

Objectives and contribution

Other surveys provide different, and sometimes more detailed, views over the field. In Ref.[1], Hoffmann et al. propose *desirable properties* for decomposition methods and analyze in details different graph-based methods w.r.t. these properties. In Ref.[2], Hoffmann and Joan-Arinyo present a less detailed survey, however providing an important list of references to decomposition and solving algorithms and systems. In Ref.[3], Sitharam presents graph-based approaches, detailing how her system FRONTIER implements such a method, discussing the issues (constriction misclassification, solution browsing, dealing with over-constriction) of the approach and proposing new techniques to handle them.

However, solving and decomposition phases are often mingled in the description of geometric constraint solvers. The comparison between methods is also difficult because different terminologies are used. This paper aims at a didactic, synthetic and homogeneous presentation of the existing decomposition algorithms. General defi-

nitions introduce the required concepts in Section 2. Based on only a few common data structures, in particular graphs, our presentation highlights the underlying operating principles, i.e., the ideas behind the techniques.

Thus, instead of a linear catalog of methods, we propose a classification of the methods in four categories and illustrate them with examples (Sections 3 to 6). The homogeneous presentation highlights advantages and drawbacks of the methods, and allows us to compare them (Section 7). Section 8 provides guidelines to meet industrial requirements. Finally, Section 9 presents a promising numerical method, called the witness configuration method, that handles several pathological cases.

2. Definitions

This section defines the basic notions and properties necessary to comprehend decomposition methods.

2.1. *Constraint systems*

A constraint system describes a set of properties (the constraints) that must be satisfied by a set of variables. The description respects syntactic rules and the variables have to take their values in specific sets. Constraint systems have thus two sides: syntactic and semantic.

Definition 1. Given a language with a semantics, a **constraint system** is a triple $S = (C, X, A)$, where C is a set of *constraints* (terms of the language), X is a set of variables and A is a set of parameters.

In *geometric* constraint systems, the variables are also called *geometric entities*. We distinguish the variables from the parameters. The variables are the unknowns to be determined while the parameters are provided, for instance, by a simulation or the user. Consider the system below:

$$\begin{cases} c_1 : dis(A, \delta_1) = 0 & c_7 : ang(\delta_1, \delta_2) = 50 \\ c_2 : dis(A, \delta_2) = 0 & c_8 : ang(\delta_1, \delta_3) = \alpha \\ c_3 : dis(B, \delta_1) = 0 & c_9 : dis(B, D) = 5 \\ c_4 : dis(B, \delta_3) = 0 & c_{10} : dis(C, D) = 6 \\ c_5 : dis(C, \delta_2) = 0 & c_{11} : dis(D, \delta_3) = d \\ c_6 : dis(C, \delta_3) = 0 \end{cases}$$

This constraint system is composed of 11 constraints (c_1 to c_{11}) and 7 variables (A to δ_3). If the system corresponds to the sketch in Fig. 1, the 2 parameters α and d are respectively set to 30 and 3. Its description language implies several types (e.g., `Point`, `Angle`) and typed functional/predicative symbols (e.g., `dis(Point,Point):Length`, `ang(Line,Line):Angle`). This syntax has two different meanings, resulting in two different constraint systems: in 2D, a system called Geo_2; in 3D, a system called Geo_3.

The purpose of a constraint system is to encode in an elegant and concise manner a set of specific assignments for its variables, the so-called solutions.

Definition 2. Let $S = (C, X, A)$ be a constraint system and let θ_A be a valuation of the parameters in A. A **solution** to S is a valuation θ_X of the variables in X such that every predicate in C is true. The set of solutions to S is denoted by $Sol(S)$.

The values of the parameters are generally fixed before a system is solved. In our example, α and d could be fixed using the dimensioned sketch in Fig. 1.

The notion of *subsystem* is paramount in decomposition methods since they are all based on the identification of solvable subparts.

Definition 3. Let $S = (C, X, A)$ be a constraint system. A **subsystem** S' of S is a constraint system $S' = (C', X', A')$, where $C' \subseteq C$, and X' and A' are respectively the variables and the parameters related to C'.

Being a subsystem is a syntactic property that does not depend on the interpretation of the constraint system. A straightforward property states that $Sol(S') \supseteq Sol(S)|_{X'}$.

2.2. *Representation of geometric constraint systems*

The presented methods operate with different representations of geometric entities and systems.

Representation of geometric constraints and entities

Several methods use a (full) geometric description of entities. They manage geometric entities, such as lines, points, planes and circles, that must satisfy geometric constraints such as distances, angles, etc.

Other algorithms work at the equational level. Geometric constraint systems are represented by systems of equations over the reals using a modeling of the geometric entities and constraints. For instance, a 2D point can be modeled by its Cartesian coordinates (x, y), and a 2D line by its polar coordinates (d, a). A point-point distance can be modeled by the classical equation $(x_1 - x_2)^2 + (y_1 - y_2)^2 = d^2$ and a line-line angle simply by $a_2 - a_1 = \alpha$.

Finally, some methods abstract the entities by their number of *degrees of freedom* (DOFs). For instance, these methods do not distinguish points and lines in 2D because both have 2 DOFs. They are often more general but less reliable.

The number of **DOFs** of a geometric entity is the number of independent coordinates used to represent it. It is equal to 2 for points and lines in 2D. The number of **DOFs** of a geometric constraint is the number of independent equations needed to represent it. For instance, angle or distance constraints have 1 DOF in 2D and in 3D. A parallelism between two planes has 2 DOFs in 3D. In the following, we denote by $dof(i)$ the number of DOFs of an entity or a constraint i.

Representation of systems of geometric constraints

Only a few decomposition methods manipulate directly the constraint system by using rewriting rules. Most of them use a graph-based abstraction of the system.

Definition 4. Let $S = (C, X, A)$ be a constraint system. Its **constraint graph**, denoted by $G_S = (V, E)$, is an undirected graph where $V = X$ (every variable in S is a vertex in G_S) and $E = C$ (every constraint in S is an edge in G_S).

Figure 2–left shows a constraint graph representing Geo_2. The entities (points and lines) are the vertexes, the constraints (distances and angles) are the edges.

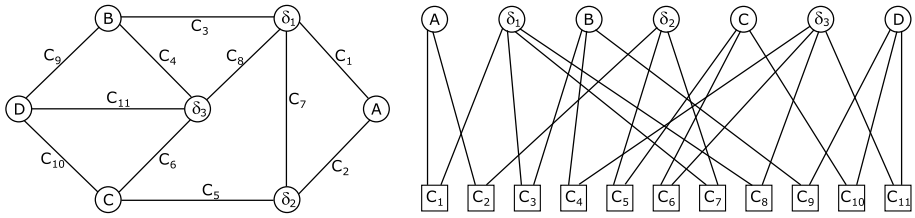

Fig. 2. Constraint graph representing Geo_2.

When the constraints are not binary (e.g., angles between triple of points), this graph includes hyper-edges. Instead of a hypergraph, one commonly uses a *bipartite constraint graph* where both the constraints and the variables are vertexes; an edge connects each constraint to each entity it constrains (see Fig. 2–right).

The methods working at the equational level operate with an **equation graph** the vertexes of which are equations and entities coordinates (also called variables). The equation graph of Geo_2 is depicted in Fig. 3. This graph can be seen as an expanded version of the bipartite constraint graph in Fig. 2–right.

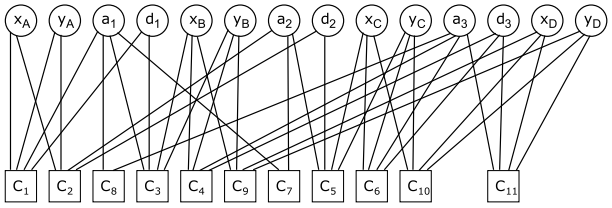

Fig. 3. Equation graph representing Geo_2.

2.3. *Constriction*

When studying a constraint system, the dimension of its solution space is important: whether the set of solutions is empty, finite or infinite changes the appropriate way to solve the problem. The notion of *constriction* captures this property.

Definition 5. (Constriction) A constraint system S is *well-constrained* if $Sol(S)$ is finite, *over-constrained* if $Sol(S) = \emptyset$ and *under-constrained* if $Sol(S)$ is infinite.

Geometric constraint systems are generally under-constrained, like Geo_2 and Geo_3, but their solutions are often identical modulo a geometric transformation (e.g., translation, rotation). Systems with such solutions are called *invariant*.

Definition 6. (Invariance) Let $S = (C, X, A)$ be a constraint system, Θ_X be the set of all possible valuations of X. Let φ be a bijection from Θ_X onto itself. S is **invariant** by φ iff $s \in Sol(S) \Leftrightarrow \varphi(s) \in Sol(S)$. Given a group G of bijections from Θ_X onto itself, S is **G-invariant** iff it is invariant by every $\varphi \in G$.

We denote $\mathbb{I} = \{id\}$ the trivial group reduced to the identity bijection. We denote $\mathbb{D}$ ($\mathbb{D}_2$ in 2D, $\mathbb{D}_3$ in 3D) the group of direct isometries. Geo_2 is $\mathbb{D}_2$-invariant and Geo_3 is $\mathbb{D}_3$-invariant. The interesting constraint systems considered in CAD are often $\mathbb{I}$-under-constrained while being $\mathbb{D}$-invariant. Hence, it is worth considering constriction modulo invariance groups.

Definition 7. (Constriction modulo a group or G-constriction) Given a constraint system S and a group G of bijections from Θ_X onto itself, S is *well-constrained modulo G* (or *G-well-constrained*) iff the orbit set $Sol(S)/G$ is finite. It is G-over-constrained iff $Sol(S)/G$ is empty, and G-under-constrained iff $Sol(S)/G$ is infinite.

For example, Geo_2 is $\mathbb{D}_2$-well-constrained while Geo_3 is $\mathbb{D}_3$-under-constrained since the quadrangle $ABCD$ can be folded continuously along line δ_3 in 3D. Systems of equations yielding a 0-dimensional solution space (i.e., having a finite number of solutions) are $\mathbb{I}$-well-constrained. $\mathbb{D}$-constriction is often referred to as *rigidity*.

The usual way to express (and to find) the solution set of an under-constrained system is to give a finite set of particular solutions $\{f_1, \ldots, f_k\}$ and an invariance group G. For instance, Geo_2 yields (generally) 2 particular solutions modulo $\mathbb{D}_2$, say f_1 and f_2, and $Sol(Geo_2) = \{\varphi(f_1)|\varphi \in \mathbb{D}_2\} \cup \{\varphi(f_2)|\varphi \in \mathbb{D}_2\}$.

2.4. *Approximate characterizations of constriction*

A crucial step in a decomposition method is to determine whether a candidate component is well-constrained (sometimes modulo a group G) or not. The constriction depends on the number of solutions. However, computing all the solutions is intractable, and would render the decomposition useless. Hence, approximate characterizations that can be checked in polynomial time are frequently used.

Pattern-based characterization

Several methods use a repertoire of known well-constrained systems. Then, checking whether a component is well-constrained amounts to matching it to a system in the repertoire. These systems are in the form of either predefined patterns or construction rules. $Pattern_1 = (C = \{dis(P, L) = k\},\ X = \{Line\ L,\ Point\ P\},\ A = \{Length\ k\})$ and $Pattern_2 = (C = \{dis(P_1, P_3) = l, dis(P_2, P_3) = k\},\ X = \{Point\ P_3\}, A = \{Point\ P_1,\ Point\ P_2,\ Length\ l,\ Length\ k\})$ are two such patterns. The first one matches $\mathbb{D}$-well-constrained components in 2D and 3D. The second one matches $\mathbb{I}$-well-constrained components in 2D only.

The patterns can also be implemented by geometric rules. For instance, $Pattern_2$ corresponds to two applications of the geometric rule *"if point P and length m are known, and $dis(P, Q) = m$, then point Q is on a determined circle C"*, and one application of the rule *"if point P is on two known independent loci $L1$ and $L2$, then P is determined"*. The rule version follows a logical approach and makes more explicit the construction associated to the corresponding pattern, yielding self-explanatory decompositions.

Pattern/rule-based characterizations use all the information available in a constraint system, i.e., its syntax and its semantics. That is why they often work directly on the system and not on an abstraction of it. This generally makes them correct, that is, a pattern-matched component is usually really well-constrained. Unfortunately, pattern-based characterizations are necessarily incomplete because no finite repertoire of patterns can cover every well-constrained component in the general case, as stated by the following proposition.

Proposition 1. *In 2D and 3D, there exist irreducible $\mathbb{D}$-well-constrained systems of arbitrary size.*

In Ref.[4], page 2[7], such systems are composed of constraints with 1 DOF (e.g., distances) and entities with 3 DOFs (resp. 6 DOFs). They are generated by simple construction steps similar to Henneberg constructions.[5]

DOF-based approximations

A well-known characterization of constriction is based on the DOF abstraction of the geometric constraints and entities. It is often called *structural rigidity* because, for several decomposition methods using this characterization, the system is abstracted by a constraint graph in which vertexes (i.e., entities) and edges (i.e., constraints) are labeled by their respective DOFs.[6]

Definition 8. (Structural G-constriction) Let $S = (C, X, A)$ be a constraint system. Let G be an invariance group of dimension $\mathcal{D}$.

The system S is *structurally G-over-constrained* iff there exists a subsystem $S' = (C', X', A')$ of S such that $\sum_{x \in X'} dof(x) - \sum_{c \in C'} dof(c) < \mathcal{D}$.

The system S is *structurally G-well-constrained* iff it is not structurally G-over-constrained and $\sum_{x \in X} dof(x) - \sum_{c \in C} dof(c) = \mathcal{D}$.

The system S is *structurally G-under-constrained* iff it is not structurally G-over-constrained and $\sum_{x \in X} dof(x) - \sum_{c \in C} dof(c) > \mathcal{D}$.

Hence, $\mathbb{I}$-constriction can be checked using $\mathcal{D} = 0$. $\mathbb{D}_2$-constriction (resp. $\mathbb{D}_3$-constriction) can be checked using $\mathcal{D} = 3$ (resp. $\mathcal{D} = 6$) which is the number of independent displacements in 2D (resp. 3D).

This characterization derives from Laman's theorem which characterizes the rigidity of *bar frameworks*, i.e., geometric constraint systems composed of points constrained by (generic) distances, systems much studied in the Rigidity Theory.[7]

Theorem 1. *(Laman, 1970) A constraint system in the 2D plane composed of n points linked by m generic distances is rigid if and only if $2 \times n - m = 3$ and for any subsystem composed of n' points and m' distances, $2 \times n' - m' \geq 3$.*

This theorem yielded several polynomial time algorithms.[8,9,10] More complicated combinatorial properties have been proposed to take into account constraints such as directions of pairs of points in 2D.[11] Unfortunately, except for 2D bar frameworks and similar systems, the structural rigidity does not imply the (geometric) rigidity. It is only an approximate characterization of it. A famous counter-example is reported in Fig. 4, where segments represent point-point distances in 3D. This system is structurally $\mathbb{D}_3$-well-constrained but actually $\mathbb{D}_3$-over-constrained[b].

Another combinatorial characterization of rigidity for 3D bar frameworks, called module-rigidity, has been conjectured by Sitharam and Zhou.[12] It can deal with

[b]This system is over-constrained in the generic case, if the two bananas have arbitrary heights. In the singular case in which the two bananas have the same height, the system is under-constrained since the two "bananas" can fold continuously along the line passing through their extremities...

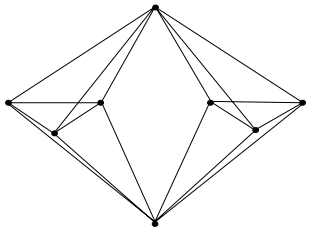

Fig. 4. The double-banana famous counter-example.

systems such as the double-banana. Module-rigidity is achievable by a polynomial algorithm and has been implemented in the FRONTIER solver.[13]

Another strong limitation of the structural characterization of rigidity ($\mathbb{D}$-constriction) is that it does not handle correctly *projective* and other *boolean* constraints such as incidences and parallelisms. However, these constraints are widely used in CAD systems. Jermann et al. introduced the notion of *degree of rigidity* which replaces, in the structural constriction, the dimension $\mathcal{D}$ of the invariance group G by a value depending on these constraints.[14]

Graph-based approximations

Another well-known approximate characterization of rigidity also derives from the Rigidity Theory community.[7] It relies on a *connectivity analysis* applied to the constraint graph of a system.

For general 2D geometric constraint systems, a necessary condition for rigidity derives from a corollary of Theorem 2.5 in Ref. [15] which states that the constraint graph must be "at most" triconnected, that is, it must contain no 4-connected subgraph. However, this condition is not sufficient. For instance, the complete graph composed of 4 points linked by 6 distances in 2D is triconnected but it is $\mathbb{D}_2$-over-constrained.

Similar necessary conditions have been proposed for 3D systems with points, lines and planes.[15] $\mathbb{D}_3$-well-constrained geometric systems composed of points and planes must be "at most" 5-connected. $\mathbb{D}_3$-well-constrained geometric systems composed of points, lines and planes must be "at most" 7-connected.

2.5. *Decomposition*

Decomposition methods transform a constraint system into a set of small solvable subsystems. To obtain the solutions of the constraint system, the (sub)solutions of its subsystems are computed and then assembled.

Definition 9. A **decomposition** of a constraint system S is a triple $(\{S_1, \ldots, S_n\}, \prec, \uplus)$ where:

- Each S_i is a G-well-constrained constraint system called a *component*.
- $\prec$ is a *partial order* for the solving of the S_is.
- $\uplus$ is a solution *assembling operator* such that if f_i is a solution of S_i, $\uplus(f_1, \ldots, f_n)$ is either an empty assignment (the f_is are incompatible) or a solution of S.

The components of a decomposition are in general subsystems of the original system[c].

The partial order expresses dependences between the components: S_j depends on S_i (noted $S_i \prec S_j$) if S_j becomes G-well-constrained only once a solution of S_i is given. These dependences may derive from variables shared by components: among the components sharing a given variable, one will compute its value and the other ones will use this value as a parameter for their own solving (see example below).

The assembling operator depends on the group G under which the components are well-constrained. If $G = \mathbb{I}$ (i.e., no invariance), then the assembling amounts to simply concatenating the solutions of the components into a solution of the original problem. When $G = \mathbb{D}_2$ or $\mathbb{D}_3$, then it amounts to computing direct isometries which make coincide the variables shared by components. The assembling operator is applied in an order related to the partial order $\prec$.

For example, Geo_2 can be decomposed as follows:

- Components:
  - $S_1 = (\{dis(B,\delta_1) = 0, dis(B,\delta_3) = 0, dis(C,\delta_3) = 0, ang(\delta_1,\delta_3) = 60, dis(B,D) = 5, dis(C,D) = 5, dis(D,\delta_3) = 3\}, \{B, C, D, \delta_1, \delta_3\}, \emptyset)$
  - $S_2 = (\{dis(A,\delta'_1) = 0, dis(A,\delta_2) = 0, ang(\delta'_1,\delta_2) = 40, dis(C',\delta_2) = 0, dis(C',\delta'_1) = dis(C,\delta_1)\}, \{A, C', \delta'_1, \delta_2\}, \{C, \delta_1\})$
- Partial order: $S_1 \prec S_2$
- Assembling operator: displacement to make C coincide with C', and δ_1 with δ'_1

This decomposition contains 2 components S_1 and S_2 corresponding to those depicted in Fig. 1. S_1 is a $\mathbb{D}_2$-well-constrained component containing 7 constraints to determine 5 variables, the points B, C and D and the lines δ_1 and δ_3. S_2 solves 4 variables, the points A and C' and the lines δ'_1 and δ_3, using 5 constraints. The constraint $dis(C',\delta'_1) = dis(C,\delta_1)$ corresponds to the metric information transferred from S_1 to S_2. This constraint renders S_2 $\mathbb{D}_2$-well-constrained and ensures that assembling the solutions of S_1 and S_2 is always possible. Because of this constraint, C and δ_1 are parameters of S_2, implying the dependence $S_1 \prec S_2$.

2.6. *Decomposition methods*

Geometric constraint solvers often work in two phases: decomposition and solving. This paper only focuses on the decomposition phase. The reader will find in the literature details about the solving methods: local numerical methods,[16,17,18] continuation,[19] interval analysis,[20,21,22] symbolic computation,[23,24] ruler and compass.[25,26]

We propose to classify the existing decomposition methods into four categories with respect to the way they operate:

- the *recursive division* methods work iteratively. At each iteration, current

[c]However, some algorithms modify the original system by incorporating a metric information (extracted from some subsystems and reported in others). The corresponding components are thus not subsystems of the original problem.

components are split along assembly points, producing several subcomponents where additional constraints are introduced to maintain the consistency of the assembly points.
- the *recursive assembly* methods also work iteratively. At each iteration, they identify a component and condense it into a new set of variables and constraints in the constraint system.
- the *single-pass* methods produce all the components simultaneously. Most of these methods are based on the maximum matching theory.
- the *propagation of degrees of freedom (PDOF)* methods iteratively identify components in reverse order of their solving.

This classification is not traditional and allows us to establish methodological similarities between existing approaches.

In addition to an algorithmic description illustrated by examples, we provide the following information for each category of methods[d]:

- *Confluence*: a method is confluent if any choice during the running process leads to the same decomposition.
- *Completeness*: a method is complete if it always returns a decomposition when there exists one. In case a method is incomplete, we explicit restrictions under which it becomes complete.
- *Complexity*: the time complexity that generally depends on the size of the constraint system.
- *Correctness*: a method is correct if the obtained components are really G-well-constrained. In case a method is incorrect, we identify restrictions under which it is correct.

3. Recursive Division Approaches

Recursive division methods, also called division analysis, work by iteratively splitting the constraint system into components, themselves subject to further splitting. In 1991, Owen introduced the first approach of this kind.[27] This method handles 2D constraint systems with points and lines linked by distance and angle constraints. It uses the *constraint graph* presented in Sec. 2.2 and the *biconnection* and *triconnection* conditions introduced in Sec. 2.4.

The constraint graph G_S is decomposed at its *articulation pairs*, i.e., pairs of vertexes whose removal splits the graph into disconnected components. Given an articulation pair (v_1, v_2), G_S is split into several subgraphs $(G_1, ..., G_n)$ by duplicating v_1 and v_2 in each subgraph. An edge, called a *virtual bond*, is introduced between v_1 and v_2 in each subgraph G_i except if there is already an edge between v_1 and v_2 in G_i or if G_i is biconnected, i.e., there exist 2 distinct paths between any pair of vertexes in G_i. Virtual bonds represent constraints that fix the relative positions of the variables shared by subgraphs (see Fig. 1). An interesting property states that if S is $\mathbb{D}_2$-well-constrained, then one of the subgraphs (say G_1)

[d]A set of properties of decomposition methods has been proposed in Ref.[1], with an emphasize toward graph properties. In this paper, we prefer using more traditional algorithmic notions.

is $\mathbb{D}_2$-well-constrained and the others are $\mathbb{D}_2$-under-constrained (see Ref. [28] for a proof). Hence, a virtual bond is not needed in G_1. When constraints are already present between articulation pairs, they are simply duplicated in all the subgraphs, avoiding the use of virtual bonds.

This process is recursively repeated on each subgraph until no more splitting is possible. In the end, every remaining subgraph is either an edge, a triangle or a triconnected subgraph; they form the set of components in the decomposition. The partial order between the components is induced by the virtual bonds: first components without virtual bonds, then the others. The assembly operator has to be applied everytime a virtual bond must be determined. It consists in computing a displacement to make coincide shared objects. Figure 5 shows the application of Owen's method to Geo_2.

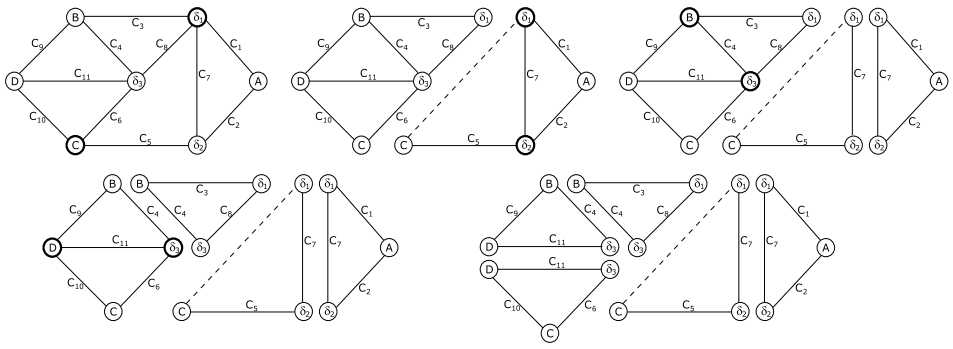

Fig. 5. Decomposition of Geo_2 by Owen's method.

This method has been extended to tackle $\mathbb{D}_2$-under-constrained systems using a completion mechanism which adds constraints in order to obtain $\mathbb{D}_2$-well-constriction.[29,30] Fudos and Hoffmann described a similar division method that is hybridized with a recursive assembly technique.[29] It is proved in Ref. [31] that their technique has the same power as Owen's method and obtains the same decomposition on $\mathbb{D}_2$-well-constrained systems. Gao and Zhang have extended the approach to 3D constraint systems made of points, planes and lines by ensuring 5-connection or 7-connection of the graph (see Sec. 2.4).[15]

Properties

These methods suffer from strong limitations, mostly because they use graph-based approximations (connectivity) of $\mathbb{D}$-constriction, rendering them correct only when these characterizations match $\mathbb{D}$-constriction (see Sec. 2.4). This also introduces another cause of incorrectness: the identified assembly pairs do not always constitute valid junctions. For instance, an articulation pair made of 2 parallel lines cannot be used to assemble components. Indeed, an infinite number of displacements can make coincide these lines, producing an infinite number of assembled solutions. Finally, this makes them incomplete in general. For instance, there exist triconnected constraint graphs that can be decomposed further. This means that the method cannot decompose all decomposable geometric constraint systems. Fig. 13–left will

show an example.

These methods run in polynomial time because they use polynomial time procedures to check k-connectivity.[32] For instance, Owen's method runs in $O(n^2)$ when the system has n constraints. They are also confluent since a choice of a given articulation pair does not eliminate another possible pair. Thus, in the end, the resulting decomposition is a unique set of components in the constraint graph.

4. Recursive Assembly Approaches

Unlike recursive division methods, recursive assembly methods adopt a bottom-up scheme. They iteratively aggregate $\mathbb{D}$-well-constrained (i.e., rigid) components into bigger ones. These methods are often referred to as *recursive rigidification*, *reduction analysis* or *decomposition-recombination* methods (DR-planners). If the constraint system contains both well-constrained and under-constrained subparts, recursive assembly methods return a maximal set of maximal well-constrained components.

4.1. *Overview*

Recursive assembly methods are all based on the same scheme:

(1) **Identification phase:** Find a rigid subsystem, called *cluster*. This phase is itself divided into two steps:
   (a) **Merge step:** Merge several variables linked by constraints into a new rigid cluster K.
   (b) **Extension step:** Perform as many *single extensions* as possible. A single extension adds a subset of variables and constraints to K such that K remains $\mathbb{D}$-well-constrained.

(2) **Contraction phase:** This phase is itself divided into two steps:
   (a) **Registration step:** Place K in the current decomposition and update the partial order.
   (b) **Replacement step:** Replace K in the system by a *representative*, i.e., a set of new variables and constraints.

These two phases are repeated until no more cluster is found. The result of the method is a decomposition where each merge or single extension represents a component. Merge components are $\mathbb{D}$-well-constrained while components obtained by extension are $\mathbb{I}$-well-constrained. Indeed, merge steps identify a new subsystem to be solved in a new local reference system, while extensions position entities relatively to an existing local reference system. The partial order between components is induced by the aggregation of the cluster representatives (to be solved before) into a larger subsystem (to be solved later). The assembling operator amounts to concatenating all the components solved within the same local reference system (a merge step + all its extensions), and then assembling by displacements the clusters whose representatives are included in other clusters.

A significant advantage is that the solving phase can be interleaved with the identification phase: once a merge step or a single extension is performed, the corresponding component can be solved immediately.

Example

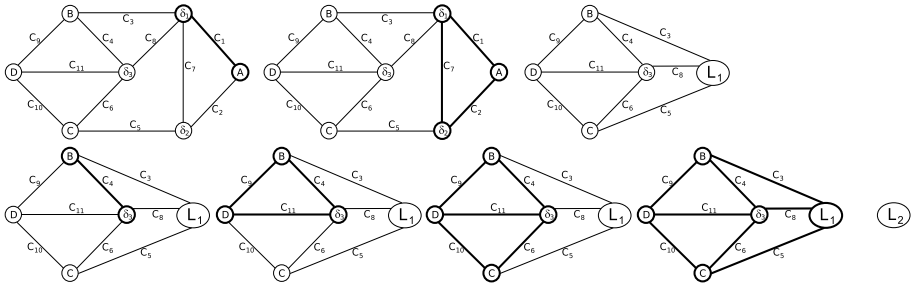

Fig. 6. Decomposition of Geo_2 by a recursive assembly method

Fig. 6 shows how a recursive assembly can decompose Geo_2. A merge step finds $A\delta_1$ as a first rigid cluster. The cluster is extended to δ_2 with constraints C_2 and C_7. The cluster $A\delta_1\delta_2$ cannot be extended anymore (no neighbor entity can be added to the cluster keeping it rigid) so that the identification phase ends. The contraction phase replaces this cluster by a single variable L_1 which represents the local reference system in which A, δ_1 and δ_2 will be determined. A second merge step identifies a rigid cluster $B\delta_3$. The cluster is extended to D, C, and finally to the cluster L_1, yielding the cluster L_2. This last extension amounts to positioning the local reference system of L_1 in the one of L_2.

Identification phase

Recursive assembly methods differ in the characterization of constriction (modulo $\mathbb{D}$ for merge step, modulo $\mathbb{I}$ for extensions steps) they employ (see Sec. 2.4). Some use DOF-based characterizations and are called *structural recursive assembly methods*. They are described in Sec. 4.2. Others use patterns or rules and are called *semantic recursive assembly methods*. They are described in Sec. 4.3.

Contraction phase

The registration phase adds the components $K = \{S_1, ..., S_n\}$ returned by the identification phase (S_1 is the result of the merge step; $S_i - i \in \{2...n\}$ – corresponds to a single extension relative to S_{i-1}) into the decomposition and updates the partial order. If S_i includes the representative of a previously identified component S', then S_i depends on S', i.e., $S' \prec S_i$.

The replacement phase removes from S all the constraints and variables of K and replaces them by a *representative*. It is either a single variable related to the whole cluster K. Or it is a subset of the variables of $S_1...S_n$ shared by constraints in the rest of the system, like the virtual bonds in Owen's method.

4.2. *Structural methods*

Hoffmann et al. in Ref. [33] introduced a flow-based algorithm to check structural rigidity in polynomial time. The merge step identifies a minimal[e] *dense* (i.e., struc-

[e]It has been proved in Ref. [34] that finding a minimum dense subsystem is an NP-hard problem. That is why Hoffmann et al. proposed to search minimal (with respect to set inclusion) dense

turally well- or over-rigid) subsystem by computing a maximum flow in a network derived from the bipartite constraint graph. The source S is linked to each constraint, and each variable is linked to the sink T. The capacities correspond to the DOFs of the constraints (arcs from the source to constraints) and to the DOFs of the variables (arcs from variables to the sink).

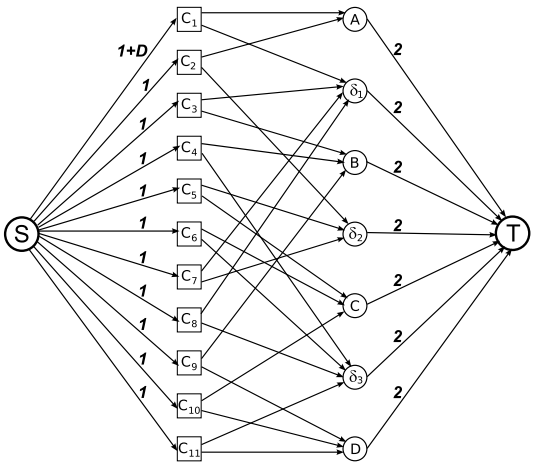

Fig. 7. Network representing Geo_2; the arc $S \rightarrow C_1$ is augmented by $\mathcal{D}$ to identify a rigid subsystem

A maximum flow in this network represents an optimal distribution of the constraints' DOFs onto the variables' DOFs. To identify rigid (or over-rigid) subsystems, the method adds an additional $\mathcal{D}$ capacity to one constraint at a time, thus fixing a local reference system onto the variables linked to the overloaded constraint.

Once a minimal cluster has been identified, the method performs an extension step which tries to include neighbor variables one by one.

This identification phase is used in two main methods by the same authors: the condensing algorithm (CA) and the frontier algorithm (FA).[1,35] The two algorithms differ only in their contracting operator. CA contracts a cluster by removing all its variables and constraints from the constraint graph. Then it introduces a new vertex which represents the local reference system of the cluster. Every constraint which was incident to a variable inside the cluster is directly transferred to this new vertex. Fig. 6 illustrates how CA decomposes Geo_2.

FA[f] contracts a cluster by removing all its *internal* variables, replacing them by a single vertex K. Variables are internal if they are not linked to any other variable outside the cluster. Otherwise, they are in the *frontier* of the cluster. The DOF of K is equal to the DOF of the subsystem it replaces. The constraints linking an internal variable i to a frontier variable f are transferred between K and f. The constraints linking frontier variables remain unchanged.

In these two variants, the partial order is induced by the integration of com-

subsystems only.

[f]The FA algorithm was called MFA in the first descriptions.

ponents' representatives into other components, yielding a *cluster tree*. When the representative of a cluster is solved in a subsequent component, a displacement is performed for assembling the components' solutions.

An extension of FA has been proposed for helping the user in case a system is over-constrained by only one equation. This algorithm provides the set of the constraints from which one can be removed to render the system well-constrained.[36]

The method has been reproduced using a maximum matching algorithm, with a contraction step similar to CA.[37] Also, Gao *et al.* in Ref. [38] have obtained an algorithm similar to FA by implementing the merge step with a weighted maximum matching algorithm like the one designed by Latham *et al.* (see Sec. 5).

Properties

Structural methods run in polynomial time, often quadratic since computing a maximum flow/matching can be done in $O(n^2)$. They are complete and correct with respect to the structural rigidity, but not with respect to the geometric rigidity (see Sec. 2.4). They are not confluent since the decomposition depends on the identification order of the clusters.

4.3. *Semantic methods*

We distinguish semantic methods according to the way they handle the identification phase: either using patterns to be identified in the system, or using rules to be triggered on a base of facts.

Pattern-based methods

Sunde (Ref. [39]) introduced a method to deal with 2D problems composed of points linked by distances and angles. Initially, each segment (point-point distance constraint, CD in short) and each pair of segments constrained in angle (CA in short) is considered a rigid subsystem. The method uses a few patterns to aggregate subsystems into bigger ones. For instance, two CAs sharing a segment, or three CDs pairwise sharing a point, can be aggregated. The method succeeds when it returns a single CD. Each pattern use corresponds to a component of the decomposition. The partial order is derived from the application order of the rules. Verroust *et al.* in Ref. [40] introduced additional patterns and proved that there exist reducible constraint systems that cannot be decomposed with this method. Joan-Arinyo *et al.* in Ref. [41] have also extended the method by considering point-lines constraints (CHs).

Fudos *et al.* in Refs. [42,43] generalized the approach by abstracting constraints and entities by their DOFs. In this method, each constraint is also initially considered a rigid subsystem, called a *cluster*. The main rule used by this method is *"3 clusters pairwise sharing a variable can be aggregated"*. This rule also appears in Sunde's method. The main difference is that CDs and CAs are not distinguished. We obtain a gain in generality but a loss in correctness. For instance, this method could assemble 3 lines linked by 3 angle constraints, which does not correspond to a $\mathbb{D}_2$-well-constrained subsystem. Applied to $\mathbb{D}_2$-well-constrained systems, this method was proved in Ref. [31] to have the same capabilities as Owen's method (see

Sec. 3). It was also combined with an equational decomposition method (see Sec. 5) in order to handle mixed algebraic-geometric constraint systems.[44,41]

Kramer in Ref. [45] proposed a similar approach to solve kinematics problems in 2D and 3D. It uses an extensive repertoire of construction steps that position one mechanical piece relatively to already known ones through both a DOF case-based reasoning (where translational DOFs from rotational ones are distinguished) and applications of the Grübler formula,[46] a property used in the theory of mechanisms and close to the structural rigidity. This method, like Fudos *et al.*'s one, mixes a structural approach and a semantic one.

Gao et al. have proposed a method using loci determination which is oriented toward 2D problems, and presented as a generalization of Sunde's approach.[47]

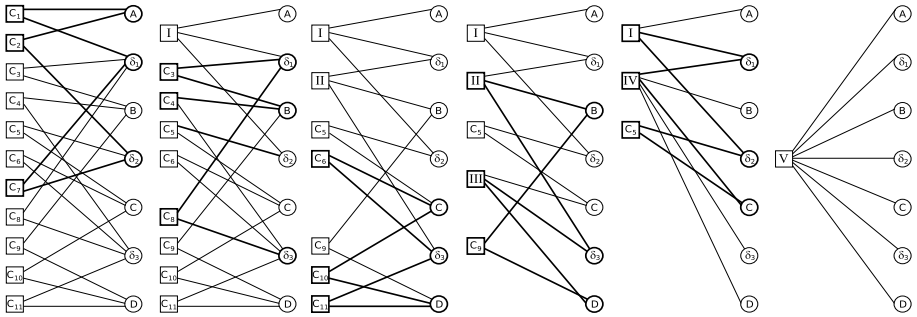

Fig. 8. Decomposition of Geo_2 computed by Sunde-like methods

Example: Figure 8 illustrates the use of Sunde's method on Geo_2. The different graphs represent the iterations of the algorithm. Initially, each constraint is considered to induce a rigid subsystem. Then, a rule identifies 3 subsystems (1, 2 and 7, in bold in the first graph) pairwise sharing a variable (A, δ_1 and δ_2). The contraction results in the second step of the figure: the three subsystems are aggregated into a single one numbered I. The following iterations use the same pattern until all the system is aggregated into a single component. Note that once a subsystem has been identified, it becomes part of a subsequent aggregation. The partial order comes from the aggregation process: $1 \prec I$, $2 \prec I$, $7 \prec I$, $3 \prec II$, $4 \prec II$, ..., $I \prec V$, $IV \prec V$ and $5 \prec V$.

Properties: Pattern-based methods are polynomial time in the size s of the biggest pattern, since a pattern has to be compared at worst with all the subsystems of size s. They are complete with respect to their repertoire of patterns (see Sec. 2.4). They are in general correct when the patterns are not DOF-based and can thus identify singular cases. Patterns can be applied in different orders, making these methods not confluent.

Knowledge-based methods

Knowledge-based methods derive from the artificial intelligence community. Seminal works were performed in the educational domain,[48,49] and then extended to

CAD.[50] The key idea consists in formalizing the geometric reasoning by a multi-typed language and a set of axioms.[51] This formalization is implemented by a rule-based solver with a forward-chaining inference system. These methods often lead to robust solvers able to give explanations in case of failure.

Mathis et al. in Ref. [28] introduced a knowledge-based method, taking $\mathbb{D}_2$-invariance into account, composed of:

- *Agents* which are mainly rule-based solvers devoted to subsystems discovery and solving. Each agent execution produces a component of the decomposition.
- A mechanism that injects a metric information (constraint) deduced from the solved subsystems into the remaining system (replacement step).

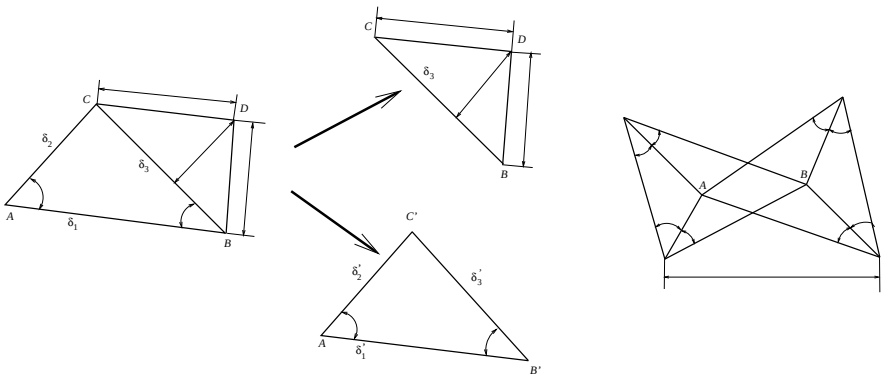

Fig. 9. **Left:** Decomposition of Geo_2 using multigroup invariance. **Right:** A constraint system decomposable by multigroup approaches only.

This framework has been extended to take other invariance groups into account, like the similarity group,[52] and even several invariances simultaneously.[53] For instance, Geo_2 could be decomposed into a similarity invariant subsystem S_1 and a rigid (displacement invariant) subsystem S_2 as illustrated in Fig. 9–left. S_1 is solved modulo the similarities, i.e., it respects the angle constraints but can still be scaled and displaced. S_2 is solved modulo the displacements, i.e., in a local reference system. The assembling operator computes the similarity (displacement + scaling) to be applied to the solutions of S_1 such that points B and C coincide in S_1 and S_2. Using the multigroup invariance allows us to decompose systems that cannot be decomposed by traditional methods, as shown in Fig. 9–right.

Properties: Knowledge-based methods are very similar to pattern-based ones, except they adopt an axiomatic approach for the decomposition phase. They run in polynomial time in general, with an exponent in the size of the biggest rule. However, some of these methods create intermediary variables, possibly leading to infinite inference chains. Knowledge-based methods are complete with respect to their repertoire of rules (see Sec. 2.4). Like pattern-based methods, knowledge-based ones are generally correct but not confluent. Finally, existing knowledge-based methods are limited to 2D systems.

5. Single-pass Approaches

Single-pass methods compute a decomposition in a single iteration: all the components are produced simultaneously. They work at the equational level and are thus not restricted to geometric constraints. Applying a *maximum matching* to the equation graph, they identify structurally $\mathbb{I}$-well-constrained components (see Sec. 2.4). In case the whole system is structurally $\mathbb{I}$-well-constrained, the decomposition into irreducible components is unique. Otherwise one obtains a coarse canonical decomposition into three subparts: an over-, a well- and an under-constrained subparts.

5.1. *Principle*

Serrano proposed the first matching-based solver for systems of nonlinear equations appearing in general design problems.[54] Since then, the method has been generalized and follows these steps:

(1) Computation of a maximum matching of the equation graph.
(2) Computation of a Dulmage & Mendelsohn (D&M) *coarse* decomposition which divides the equation graph into three canonical parts: $\mathbb{I}$-well-constrained, $\mathbb{I}$-over-constrained and $\mathbb{I}$-under-constrained.
(3) The $\mathbb{I}$-over- and $\mathbb{I}$-under-constrained parts are either ignored or transferred into the $\mathbb{I}$-well-constrained one by a specific treatment.
(4) Computation of a *fine* decomposition of the $\mathbb{I}$-well-constrained part. The components and the partial order between them are then determined.

Computation of a maximum matching

The decomposition is based on a maximum matching of the equation graph.

Definition 10. A **matching** of a graph is a subset of its edges, such that no two edges in the matching share a vertex. A vertex is *saturated* if it belongs to an edge of the matching. A *perfect* matching saturates all the vertexes. A matching is *maximum* iff no other matching of the same graph contains more edges.

Intuitively, an edge (c, v) in the matching means that the equation c computes the variable v. Well-known polynomial algorithms compute a maximum matching in a bipartite graph.[55] The obtained matching yields an implicit orientation to the equation graph: each edge in the matching is oriented in both directions, while edges outside the matching are oriented from the variables to the equations. An example is shown on Fig. 10[g].

A decomposition of a maximum matching into strongly connected components (i.e., cycles) is obtained by a linear depth first search algorithm by Tarjan.[56]

Definition 11. A directed graph is **strongly connected** iff for any two vertexes x and y in G, there exists a directed path from x to y and from y to x. The strongly connected components (scc) of G are its maximal strongly connected subgraphs.

[g]The point D and the direction a_3 of the line δ_3 are fixed by the unary constraints f_1, f_2, f_3 to render the whole system $\mathbb{I}$-well-constrained.

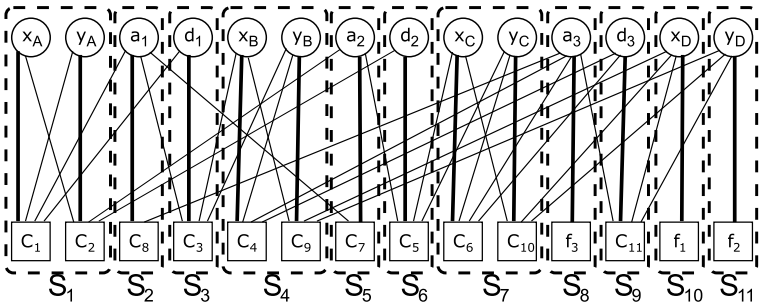

Fig. 10. D&M and scc decompositions of Geo_2. Edges in the matching are bold-faced. The components are represented by dashed hyper-edges.

Arcs *between* the sccs yield the partial order of the decomposition. In our example, this yields: $S_{11} \prec S_9$, $S_{11} \prec S_7$, $S_{11} \prec S_4$, ..., $S_2 \prec S_5$, $S_2 \prec S_3$, $S_2 \prec S_1$.

Before a subsystem is solved, the variables outside the corresponding scc are replaced by the values that have been computed in previous subsystems. For example, for solving the last subsystem S_1, the variables a_1, d_1, a_2 and d_2 must be replaced by their values computed in subsystems S_2, S_3, S_5 and S_6. The geometric interpretation of this computation is that the point A is placed at the intersection of lines δ_1 and δ_2 which must be determined first.

When the whole system lies in the well-constrained part, i.e., it admits a perfect matching, it can be decomposed into sccs as described above. The following theorem ensures that this decomposition is independent from the matching used.[57]

Theorem 2. *(König, 1916) Every perfect matching of a bipartite graph leads to a unique decomposition into strongly connected components (i.e., irreducible $\mathbb{I}$-well-constrained subsystems).*

Dulmage and Mendelsohn (D&M) decomposition

If the obtained maximum matching is not perfect, some equations and/or some variables are not saturated. Unsaturated variables are not determined, i.e., they induce a structurally under-constrained subsystem. Unsaturated equations are not taken into account and induce a structurally over-constrained subsystem. Dulmage and Mendelsohn have proposed a decomposition approach which applies to large sparse systems of linear or non linear equations.[58] Figure 11 shows an example.

Theorem 3. *(Dulmage and Mendelsohn, 1958) Let G be an equation graph. Any maximum matching of G gives a canonic partition of the vertexes in G into three disjoint subsets: the under-constrained part U_G, the over-constrained part O_G and the well-constrained part W_G.*

The existence of non-empty subparts O_G or U_G in a constraint system generally represents an error which can be returned to the user or automatically repaired.

5.2. *Properties*

The D&M and scc decompositions can be computed in polynomial time, basically quadratic: computing a maximum matching is done in quadratic time (assuming the

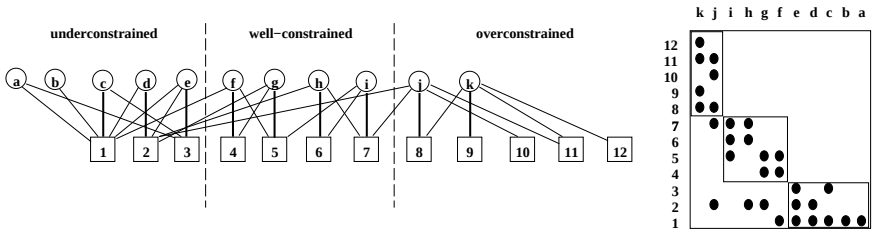

Fig. 11. **Left:** The three parts found by D&M's decomposition. The equation graph contains equations 1...12 and variables $a...k$. **Right:** Equivalent matrix representation (the square in the middle corresponds to the well-constrained part).

number of equations is close to the number of variables, and the arity of equations is bounded by a constant). Identifying the three parts and the sccs requires only graph walks performed in linear time.

The method is complete and correct with respect to the structural $\mathbb{I}$-constriction. D&M's decomposition and the fine decomposition of the well-constrained part into sccs are confluent. However, the handling of the over- and under- constrained parts may induce some changes in the resulting decomposition.

5.3. *Difficulties due to the application to geometry*

Geometric constraint systems are often $\mathbb{D}$-well-constrained. Thus, corresponding equation systems are generally $\mathbb{I}$-under-constrained, and the whole equation graph falls into the under-constrained part. Because König's canonicity result does not hold anymore in this case, the scc decomposition then depends on the computed maximum matching, which leads to two major flaws:

- Some matchings lead to smaller subsystems than others. Fig. 13–right will show a non optimal decomposition of Geo_2.
- Some matchings lead to geometrically incorrect decompositions. For instance, consider that Geo_2 is rendered well-constrained by fixing x_B, x_C and x_D. This results in a single scc (no decomposition) which is structurally $\mathbb{I}$-well-constrained while it is $\mathbb{I}$-over-constrained in reality: the prescribed distances BD and CD cannot be satisfied for every value of their abscissa.

To decompose rigid geometric constraint systems in a correct manner, Hendrickson proposed to fix the geometric system in a coordinate system, removing thus the 3 degrees in freedom in excess in 2D (6 in 3D).[10] When the geometric system is a 2D bar framework, every edge (representing a point-point distance in the constraint graph) is pinned in the plane by adding 3 similar edges, and a maximum matching is computed on the modified system. The system is rigid iff all the combinations yield a perfect matching. This method is performed incrementally, and only the first maximum matching has to be computed from scratch. The others are obtained in linear time by updating the previous one.

Latham and Middleditch proposed to replace the maximum matching computation by a weighted maximum matching one.[59] The constraint graph is weighted by the degrees of freedom of the variables and the constraints. The advantage is that it is performed directly on the constraint graph of the geometric system, avoiding

the need to translate it into an equation system. This method was also able to add (resp. remove) constraints in an ad-hoc manner in case the system is under-rigid (resp. over-rigid). This algorithm has been used in several alternatives to the structural recursive assembly method due to Hoffmann *et al.* (see Sec. 4.2).[37,38]

6. Propagation of Degrees of Freedom Approaches (PDOF)

Initially implemented by Sutherland in his graphical tool Sketchpad,[60] PDOF[h] has been first used in local propagation solvers for handling interactive constraint systems, such as graphical layout systems or user interfaces.[61,62] As we will see, pattern-based PDOF algorithms overcome the main drawbacks of the maximum matching approach when tackling structurally $\mathbb{I}$-under-constrained systems.

6.1. *Description of a generic PDOF*

The PDOF algorithm has first been applied at the equational level, that is, it works on an equation graph.[60,61,62] Several approaches in geometry directly work with the (geometric) constraint graph.[63,64,65,66] In both cases, the principle of PDOF is to iterate the following steps:

(1) select a **free** $\mathbb{I}$-well-constrained subsystem S' from the equation/constraint graph G,
(2) remove from G vertexes and edges corresponding to S'.

The algorithm stops when the graph is empty or when it contains no more free well-constrained subsystem. A subsystem is *free* if its variables appear only in the equations of the subsystem. Thus, solving a free subsystem cannot violate equations not in S', i.e., no future component will depend on S'. Each free subsystem identified by PDOF is a component of the decomposition. The partial order is obtained by considering variable dependences: a component S_j depends on a component S_i if S_i computes a variable which is involved in an equation of S_j.[i] The assembling operator amounts to concatenating the components solutions.

In case the constraint system is under-constrained, when all the equations have been removed from the equation graph, a set of variables (called *input parameters*) remains. Hence, PDOF is also a procedure that determines a set of input parameters to be fixed for rendering $\mathbb{I}$-well-constrained the system.

We illustrate how PDOF computes a decomposition and then we provide details about the existing algorithms that are based on the generic algorithm above.

6.2. *Example*

Fig. 12 shows how PDOF decomposes Geo_2 (among other alternatives). It selects free subsystems in the order M_1 (computing point A at the intersection of δ_1 and δ_2), M_3, M_6, M_4, M_5, M_2, M_7, M_9. It selects first M_1 because M_1 is free, that is, variables x_A and y_A are connected only to C_1 and C_2 which belong to M_1. Once M_1 is selected and removed from the equation graph, M_3 and M_6 become free and

[h]PDOF stands for Propagation of Degrees of Freedom.
[i]Note that a total order can be trivially obtained by reversing the identification order: the first identified subsystem is solved last.

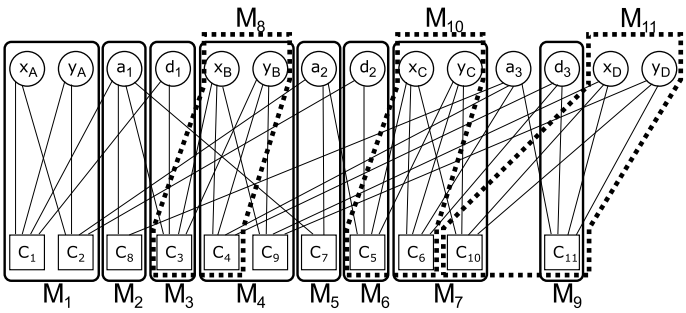

Fig. 12. Equation graph of Geo_2 with subsystems $M_1 ... M_{11}$ represented by hyper-edges corresponding to patterns in a dictionary (for the sake of clarity, all the subsystems have not been drawn). Several subsystems, namely the hyper-edges in plain lines, are selected by PDOF to decompose the system.

can be selected next, and so on. At the end, the remaining variables a_3, x_D and y_D constitute a set of input parameters. Selected subsystems are solved in reverse order (i.e., from M_9 to M_1).

6.3. *Algorithms based on the generic PDOF*

The basic `PDOF` algorithm, derived from local propagation, iteratively selects a free subsystem of size 1. This gives a triangular form to the equation-variable dependence matrix. However, it is often not possible to triangulate a geometric constraint system with only subsystems of size 1.

`OpenPlan`: a purely structural PDOF

Like Single-pass methods, `OpenPlan` works at the equational level with no geometric information.[67] It tries to select a free structurally $\mathbb{I}$-well-constrained subsystem *of minimum size* at each step. `OpenPlan` returns the best (in terms of size of the largest component) decomposition that can be obtained by a maximum matching applied to an under-constrained system. However, the problem of finding a free subsystem of smallest size in a structurally $\mathbb{I}$-under-constraint system is NP-hard. Indeed, it is the dual of the NP-hard *minimum dense* problem (searching for a structurally rigid subsystem of minimum size). [34] That is why a heuristic version of `OpenPlan` also uses a maximum matching of the equation graph to find a small, but not necessarily the smallest, free subsystem.[67]

Pattern-based PDOFs

As opposed to the standard PDOF, pattern-based PDOFs can identify free $\mathbb{I}$-well-constrained subsystems of bigger size. Pattern-based PDOFs make a bridge between the equational and the geometric levels by using a dictionary of subsystem patterns. These patterns correspond to geometric construction steps. In the example above, the pattern *"point to be placed at the intersection of two known lines"* in the dictionary corresponds to several subsystems in the actual system, such as M_1 (point A intersection of δ_1, δ_2), M_8 (point B) and M_{10} (point C).

Basic pattern-based PDOFs apply the generic scheme presented above.[65,66,63] They iteratively select free subsystems (of arbitrary size) present in the dictionary.

However, it appears that these algorithms may be unable to compute a sequence of subsystems corresponding to patterns in the dictionary, even if one such sequence exists. A simple example can be found in Figure 6 of Ref. [68]. To overcome this drawback, one needs to be able to select subsystems that "overlap", that is, share equations and variables. This analysis has led to the design of *General PDOF* (`GPDOF`) that can be viewed as a robust implementation of a pattern-based PDOF.

`GPDOF` is able to compute a sequence of subsystems corresponding to patterns in the dictionary, if one such sequence exists. To be able to select subsystems that overlap, `GPDOF` requires that the equation graph be enriched in advance with hyper-edges corresponding to patterns. For instance, this preliminary enrichment phase produces the equation graph depicted in Figure 12 enriched with hyper-edges $M_1 ... M_{11}$. See Refs.[26,69] for more details on this algorithm.

6.4. *Properties*

Like Single-pass, `OpenPlan` is complete and correct, with the usual limitations of structural methods due to an incorrect constriction approximation related to redundancies or singularities. Provided that the system contains no redundant equations, `GPDOF` can always compute a sequence of subsystems present in the dictionary if one such sequence exists, but a strong limitation is of course the non exhausitivity of the dictionary.

Pattern-based PDOF methods run in polynomial time. The standard PDOF algorithm runs in $\mathrm{O}(n \times dv \times dc^2)$ while `GPDOF` runs in $\mathrm{O}(n \times dc \times dv \times r \times (g \times dc + g^2))$, where n is the number of equations, r is the maximum number of hyper-edges per equation, dc and dv are the maximum degrees of respectively an equation and a variable in the equation graph, and g is the maximum number of equations and variables involved in a hyper-edge. Note that r is $O(n^g)$, rendering the method practicable for small patterns only. In an application of `GPDOF` to constraint-based 3D scene reconstruction, large under-constrained geometric systems are decomposed in a few seconds and solved in hundredths of seconds.[26,69]

7. Comparison Between the Four Decomposition Schemes

In this section, we compare the four decomposition schemes we have presented. These approaches differ not only in their algorithmic aspects, but also in the abstraction they use. Taking these differences into account allows us to better explain their relative strengths and weaknesses.

7.1. *Recursive division versus recursive assembly*

Recursive division methods use k-connectivity as an approximation of $\mathbb{D}_2$-constriction, which renders them incorrect and incomplete in general (see Sec. 2.4). Consider the 2D constraint system composed of 12 points ($A...L$) linked by 21 distances whose constraint graph is depicted in Fig. 13–left. This graph is triconnected so that Owen's method cannot decompose it. However, Hoffmann *et al.*'s recursive assembly method can decompose it further: all the small triangles are aggregated recursively and finally the big triangles ACF and GJL are aggregated using the three distances AJ, FG and CL.

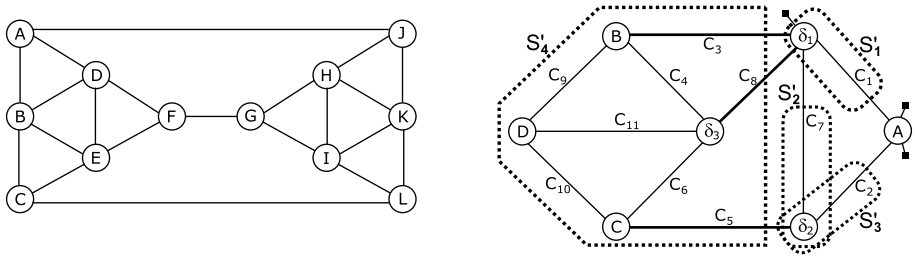

Fig. 13. **Left:** A decomposable problem that Owen's method cannot treat. **Right:** Coarse decomposition (in dashed lines) of Geo_2 by Maximum-matching.

This poor characterization of rigidity explains why recursive assembly is generally considered more powerful than recursive division. This difference is thus not really related to their respective schemes (bottom-up or top-down).

Concerning the schemes, recursive division methods can often better decompose under-constrained systems because the more under-constrained the problem is, the more articulation pairs can be found. On the contrary, because merge operators intrinsically identify well-constrained or over-constrained components, recursive assembly methods are best suited for over-constrained systems.

7.2. *Single-pass versus PDOF*

Dulmage&Mendelsohn's decomposition of single-pass methods is always interesting as a preliminary step to decompose a geometric constraint system: it identifies a structurally $\mathbb{I}$-over-constrained part O, an $\mathbb{I}$-well-constrained part W and an $\mathbb{I}$-under-constrained part U of the system.

Part O cannot be solved: it requires some specific treatment (e.g., manual debugging or automatic relaxation of constraints). Part W admits a unique decomposition (see Sec. 5.1). Part U requires a further study: if obtained by any maximum matching, the resulting decomposition often contains arbitrarily large subsystems (see Fig. 13–right). On the opposite, a pattern-based PDOF provides only meaningful components of bounded size. In addition, the more under-constrained the system, the easier PDOF will find a free subsystem.

The weakness of a pattern-based PDOF is its incompleteness due to the limited number of patterns registered in its dictionary. Moreover, redundancies often block the decomposition process because redundant equations may prevent `PDOF` from finding a free subsystem (due to these constraints in excess). Redundant equations must therefore be detected and removed before PDOF is launched.[69]

7.3. *Equational versus geometric approaches*

On one hand, equational methods (single-pass and PDOF approaches) can deal with constraint systems combining geometrical and non geometrical entities, providing a greater generality. In addition, they sometimes better decompose a system because they work with a finer grain: a single geometric entity is represented by several variables that may be solved in different components of the decomposition. For instance, in Fig 12, page 22, the variables a_1 and d_1 modeling line δ_1 appear in two components M_2 and M_3.

On the other hand, geometric decompositions (recursive division and recursive assembly approaches) generally use constriction modulo the displacements and exploit an assembling operator based on displacements. This allows them to use several times the 3 DOFs in 2D (resp. 6 in 3D) of a rigid geometric system, once for each identified rigid component. In comparison, equational methods can only aggregate $\mathbb{I}$-well-constrained components and hence can only use once these DOFs. That is why geometric decomposition methods often yield finer decompositions.

Combining the advantages of both methods is possible. Each component obtained by a geometric decomposition can sometimes be further decomposed by an equational technique that reduces the (geometric) components to several (equational) sub-components. This principle has been followed for obtaining the decomposed systems studied by Jermann *et al.*[22]

Finally, when facing $\mathbb{D}$-under-constrained systems, equational and geometric decomposition methods *repair* them differently. The former fix variables, while the latter add geometric constraints (e.g., distance constraints). The reason is that equational methods achieve $\mathbb{I}$-constriction while geometric ones achieve $\mathbb{D}$-constriction.

7.4. *Semantic versus structural methods*

Semantic methods (i.e., rule-based recursive assembly and pattern-based PDOF) cannot define all the patterns necessary to tackle every geometric constraint system (see Sec. 2.4)[j]. On the other hand, the main advantage of rule-based methods lies in their proximity to geometry. They generally use geometrically sound operations that are protected against failure cases. For instance, a guard could detect that a point cannot be computed at the intersection of two lines because they are parallel. Also, a fast solving method is often known to solve the subsystems.

In comparison, structural methods (graph-based recursive division/assembly, and single-pass) are in general complete but ensure correctness only with respect to an approximation of constriction defined by a structural property (e.g., triconnection or a DOF count).

Structural and semantic methods could complement each other, ensuring the completeness of the approach while achieving a higher reliability. Two such hybrid algorithms are based on the same principle: a structural decomposition is updated while a pattern-based decomposition is performed. Every time the decomposition process is stuck because no existing pattern is available, a subsystem is picked in the structural decomposition. In a Maximum-Matching/PDOF hybrid algorithm introduced in Ref. [68], a *leaf* (in the partial order) of a maximum matching decomposition is free and is selected as the next solvable component. In a Maximum-Matching/bottom-up hybrid algorithm introduced in Ref. [70], a *root* (in the partial order) of a maximum matching decomposition is selected.

8. Toward Real-life Requirements

In this section, we establish a list of properties that are desirable to meet real-life applicative requirements: *generality* and *reliability*. Indeed, a decomposition method

[j]However, for some particular classes of applications, for instance mechanical assemblies, such methods can be made almost complete.[45]

should be as *general* as possible in order to tackle the broadest class of systems. It should also be *reliable* in the sense it should adapt to the expectations of the user and be able to overcome situations that often make the decomposition fail (e.g., singular or dependent constraint systems).

8.1. *Generality*

Methods able to handle 2D and 3D problems can address complex CAD applications where 2D constrained sketches are used in complement to 3D constrained models. However, it is certainly unnecessary to use a complex 3D method for bar-frameworks in 2D. Also, ruler & compass techniques suffice for some applications and offer a higher reliability than more general methods. Hence, determining the required level of generality for an application is important to select the appropriate method.

Geometric decomposition methods should be able to use several invariance groups to decompose systems as finely as possible. One of the strengths of recursive assembly method comes from its combination of $\mathbb{D}$-invariance (merge step) with $\mathbb{I}$-invariance (extension step). Extension is a cheap augmentation technique resulting in small $\mathbb{I}$-well-constrained components, while a more expensive merge operation ensures the completeness of the method.

A general decomposition method should be able to handle under-constrained and over-constrained problems, at least by identifying as precisely as possible the parts responsible for the non well-constriction, and ideally offering some automatic repairing tools. Note that certain decomposition schemes are more suited to handle these cases than others. Recursive division and PDOF are poor against over-constrained systems but stronger against under-rigid ones. Conversely, recursive assembly can handle over-constrained parts but requires some adaptations to deal with under-constrained systems. Finally, single-pass methods are able to characterize the well-, under- and over-constrained parts, but the non well-constrained ones require a specific treatment.

The decomposition returned by a method could be best employed by a user for debugging purpose (e.g., identification of over-/under-constrained components) if it corresponds to the view the user has of its system. For this reason, it is generally desirable to respect a coarse decomposition induced by a high-level user's manipulation of entities (e.g., mechanical pieces). Sitharam et al. proposed to adapt graph-based recursive assembly methods to this requirement.[13]

The decomposed system should return the solution expected by the user, e.g., the solution minimizing some criterion or closest to a given sketch. This problem is referred to as *root identification* by Fudos and Hoffmann.[29] Geometric constraint solvers use heuristics to select the desired solution, or use a combinatorial process where one branch in the search tree corresponds to one (sub)solution in a component.[71,22]

Finally, it happens in several applications like robotics, CAD or structural biology that the constraint system contains not only geometric entities and constraints but also algebraic variables and equations, representing for instance physical laws, costs or energy. It is hence important to be able to handle these mixed algebraic-geometric systems.

Among the methods classified in the proposed four categories, none satisfies all these requirements simultaneously. We expect that hybrid variants between some of these algorithms should result in an increasing generality.

8.2. *Reliability*

Usual algorithmic properties (i.e., correctness, completeness, confluence and complexity) provide a first measure of the reliability of a decomposition method.

The ability of a method to return the decomposition into the smallest components is also important. However, this requirement should be balanced with considerations about performance. Indeed, achieving the finest decomposition has been proved to be NP-Hard in several cases.[34]

Decomposition methods should be aware of dependences between constraints. A subset of constraints is dependent if it is either redundant or contradictory, i.e., if it contains the hypothesis and the conclusion, or the negation of the conclusion, of a geometric theorem. Some dependences can be detected structurally, e.g., 4 points linked by 6 distances in 2D. However, many dependences are not structural but are related to the geometric nature of the entities and constraints, e.g., the double-banana 3D system (see Fig. 4). Handling dependences requires expensive automated theorem proving. This justifies the use of heuristics to handle the more common cases.

Decomposition methods should take into account the singularities. Indeed, many methods work under a genericity hypothesis and decompose systems into generically solvable components. However, it may be the case that a solution of a decomposed system lies into a singular variety, e.g., includes some unspecified collinearity or coplanarity. In this case, it happens that the generically solvable components are no more solvable. For instance, the double-banana system (see Fig. 4) is generically over-constrained but becomes under-constrained if the height of both bananas is the same. Dealing with singularities requires time-consuming algebraic computations and is generally incomplete. Also, singularities sometimes introduce dependences or transform contradictory ones into redundant ones.

Among the decomposition schemes we have listed, none appears to perfectly deal with these reliability issues. They are generally incomparable with respect to the size of the decomposition they obtain. Structural methods can detect only structural dependences, cannot distinguish between redundant and contradictory cases and does not take any singularity into account. Semantic methods can include rules and guards that check certain non-structural dependences and singularities.

However, no *a priori* decomposition method can be made completely reliable since there exist systems that have both singular and non singular solutions (see Fig. 14), so that certain error detections could only be performed *during* the solving phase. Such systems would be best handled by methods interleaving decomposition and solving phases (e.g., recursive assembly).

Although not able to cope with this last problem, the WCM presented in the next section makes a sensible step toward generality and reliability.

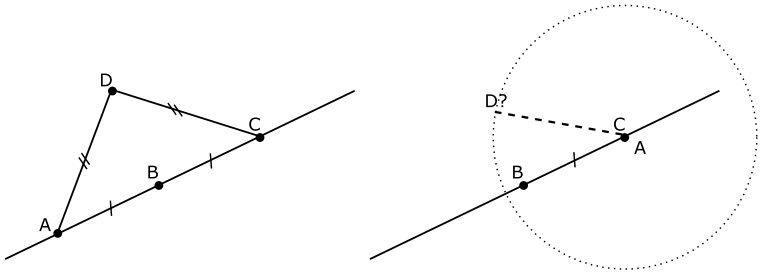

Fig. 14. A constraint system which is both well-constrained and under-constrained. An equality distance is defined between AB and BC, and also between AD and CD; Distance constraints AB and AD are also defined; A, B and C are incident to the line. One solution (left) for the position of A and C is generic while the other (right) is singular and yields an infinity of solutions for point D (it introduces a redundancy).

9. The Witness Configuration Method

We have seen that one cause of failure of decomposition methods is that they are unable to accurately characterize the rigidity property, i.e., they can identify components that are not solvable because they are either over-constrained (no solution) or under-constrained (infinitely many solutions).

To conclude this survey, we present a recent development in geometric constraint processing that provides a more general and reliable characterization of solvability: the witness configuration method (WCM).[72,73] We show that this principle can be integrated in existing decomposition methods or be the base of new ones.

9.1. *Principle of the WCM*

To determine whether a geometric constraint system $S = (C, X, A)$ is rigid, the WCM combines the following techniques:

(1) a generalization of the algebraic rigidity check for bar frameworks by infinitesimal motions computation,[74,7]
(2) the Numerical Probabilistic Method (NPM) that numerically checks algebraic properties using a probabilistic argument and random configurations,[75]
(3) a new method that generates *witness configurations* instead of random ones.[72,73]

The probabilistic rigidity check

The principle of the algebraic infinitesimal rigidity[k] check is to compute the infinitesimal motions allowed by the constraints of a geometric system S. A *basis* of these infinitesimal motions is defined by the *kernel* K of the Jacobian matrix J of the equation system representing S. If K reduces to a basis D of the infinitesimal displacements, then S is rigid; otherwise it is not rigid: if D contains one vector independent from K, S is over-rigid; else (i.e., K contains a vector independent from D) S is under-rigid.[74,7] This principle was introduced for bar frameworks but can be generalized to any geometric entities in any dimension.

[k]Infinitesimal rigidity is a first-order version of rigidity. It is a stronger property than rigidity, i.e., every infinitesimally rigid system is also rigid; the converse is not true.[76]

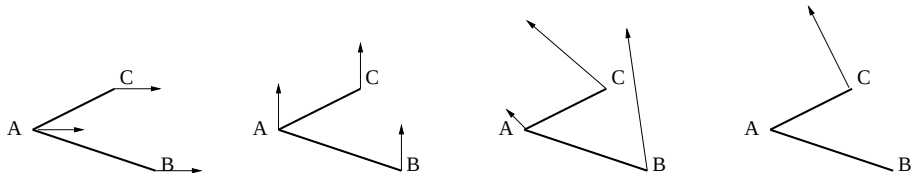

Fig. 15. An under-rigid 2D geometric constraint system

Let us illustrate the principle using the simple example S depicted in Fig. 15, composed of 3 points linked by 2 distances. In this system, assuming the variable vector is $(x_A, y_A, x_B, y_B, x_C, y_C)$, a basis K of the infinitesimal motions of S is (illustrated from left to right in Fig. 15): $\dot{X}_1 = (1, 0, 1, 0, 1, 0)$ is the infinitesimal translation in x; $\dot{X}_2 = (0, 1, 0, 1, 0, 1)$ is the infinitesimal translation in y; $\dot{X}_3 = (-y_A, x_A, -y_B, x_B, -y_C, x_C)$ is the infinitesimal rotation around the origin; $\dot{X}_4 = (0, 0, 0, 0, -y_C, x_C)$ is a last independent infinitesimal motion. $(\dot{X}_1, \dot{X}_2, \dot{X}_3)$ is also a basis D for the infinitesimal displacements allowed in 2D. Since K does not reduce to D, S is not rigid. However, K contains D and thus S is not over-rigid. Finally, since $\dot{X}_4$ is independent from D, it is a flex and S is under-rigid.

To avoid resorting to exponential computer algebra methods for computing the infinitesimal motions, the NPM method can be used.[10,75] In this case, the Jacobian matrix J of $S = (C, X, A)$ is evaluated at a randomly chosen configuration $(X, A) = (\theta_X, \theta_A)$. The kernel of $J(\theta_X, \theta_A)$ can then be computed using a simple Gaussian elimination in polynomial time. The result of this numerical computation on a sample configuration extends to the geometric constraint system using a probabilistic argument: if the point (θ_X, θ_A) is picked at random in a dense field, then the algebraic properties at this point are generically those of the algebraic system.[77]

However, this method (both algebraic and numerical flavor) applies only under a genericity assumption, e.g., null distances and null angles are not permitted. Indeed, it does not take into account the values of the parameters in A: computer algebra neglects the right part of the equation system, and the NPM uses a random (generic) assignment for the parameters. This strong assumption makes it difficult to use the method in practice since collinearities, coplanarities and other *singular* constraints are frequent in several applications (e.g., CAD).

The witness configuration

To overcome this limitation, Michelucci et al. proposed to use the NPM with a *witness configuration* instead of a random one.[73] A witness configuration of a system $S = (C, X, A)$ is a configuration $(X, A) = (\theta_X^w, \theta_A^w)$ that satisfies all the singular constraints in S, such as point/line/plane incidences. Indeed, these constraints imply that the solutions of S all lie in singular components of the configuration space, i.e., they may not have the generic properties of S. Hence, if the infinitesimal motions of S are computed using the NPM at a witness configuration instead of a random one, then the inferred constriction will hold for its solutions.

Assuming that all the singular constraints are explicitly stated by the user, a witness configuration can be computed as a solution of a reduced system S' which includes only the singular constraints of S. In theory, this problem is as complicated

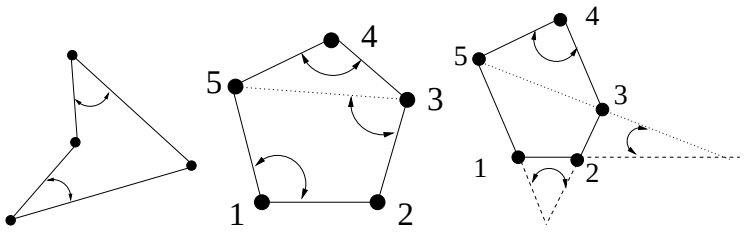

Fig. 16. Three 3D configurations (courtesy by A. Ortuzar, Dassault Systèmes).

as solving the original system S. However, in practice, the reduced system S' is highly under-constrained and can generally be solved easily. For instance, it could be composed of only point-line incidences which can be satisfied by picking any points on lines themselves randomly positioned.

Let us illustrate the complete process on the simple system S introduced in Fig. 15. Suppose that $dist(A, C) = 0$ (*singular* point-point distance) while $dist(A, B) = k$, $k > 0$ (generic point-point distance). To compute a witness configuration $(x_A^w, y_A^w, x_B^w, y_B^w, x_C^w, y_C^w, k^w)$, we can pick points A^w and B^w at random (or read them from a sketch if available): $A^w = (0, 0)$, $B^w = (3, -1)$; this determines the value of $k^w = \sqrt{10}$. Then we set C^w to the same coordinates as A^w so that the coincidence constraint is satisfied: $C^w = (0, 0)$.

Now, the basis of the infinitesimal motions of S contains only 3 vectors: $\dot{X}_1(P) = (1, 0, 1, 0, 1, 0)$ (the translation in x), $\dot{X}_2(P) = (0, 1, 0, 1, 0, 1)$ (the translation in y) and $\dot{X}_3(P) = (0, 0, 1, 3, 0, 0)$ (the rotation around A). Indeed, $\dot{X}_4$ is equal to $(0, 0, 0, 0, 0, 0)$ in this witness configuration P and is thus not independent from the three other motion vectors. Since this basis reduces to an infinitesimal displacement basis, we conclude that S is rigid at the witness configuration, and so must be all its solutions. Remark that S remains generically under-rigid.

Properties of the WCM

The WCM has some interesting properties, in particular in comparison to graph-based characterizations of rigidity:

- the WCM can be used for numerical geometric theorem proving. Indeed, if the witness configuration has some property (alignment of 3 points for instance), then this property is a consequence of the singular constraints of the system.
- the WCM can detect not only the structural dependences identified by structural methods, but also other subtle non structural ones, e.g., the double-banana configuration (see Fig. 4) and other similar 3D configurations (see Fig. 16). Michelucci et al. argue that the WCM detects all the dependences due to geometric theorems, which occurs each time the system contains the hypotheses and the conclusion (or its negation) of a geometric theorem.
- the WCM can detect bad values of the parameters in the constraints. Indeed, if a witness configuration exists and still the solver fails, i.e., the reduced system S' can be solved but not the complete system S, it is due to bad numerical

values of the parameters of the constraints, e.g., 3 distance constraints that violate the triangular inequality.

These properties make the WCM a very interesting and useful tool for modeling, solving and debugging geometric constraint systems.

9.2. *Decomposing with the WCM*

The basic step in many decomposition methods is to determine whether a part of the system is well-constrained or not, for the purpose of identifying solvable components for the decomposition. The WCM can achieve this task with a higher accuracy than typical structural approaches. Moreover, the WCM can handle more general systems than most of these approaches. Hence, the method should be able to extend the capabilities (generality and reliability) of structural methods in the four categories presented in this survey. At least, Michelucci et al. have proposed a recursive division method which uses solely the WCM.[73]

A WCM-based recursive division method

Let us call `WCM-RD` the recursive division method based on the WCM described in this section. This method operates as follows:

(1) find a set of maximal rigid subsystems $\{S_1, ..., S_k\}$ using the WCM,
(2) for each S_i, select a constraint e in S_i and apply `WCM-RD` to $S_i \setminus \{e\}$.

In the first step of this algorithm, the WCM is used to identify each maximal (w.r.t. set inclusion) rigid subsystem. For this purpose, an *anchor* A is selected at random in the system S. An anchor is a small subset of objects which fixes all the possible displacements when determined. Pairs of non coincident points in 2D, and triples of non aligned points in 3D are examples of such anchors.

To determine a maximal rigid subsystem with respect to an anchor A, the WCM computes the motions of each geometric entity o relatively to the anchor A. If the basis of these motions is empty, then o is fixed relatively to A. Once all objects fixed relatively to A have been identified, they form (with A) a maximal rigid subsystem.

To find a maximal set of maximal rigid subsystems in S, the process described above is repeated for all possible anchors in sequence.

In the second step, the `WCM-RD` method is recursively applied to every maximal rigid subsystem found at step (1) after having removed a constraint picked at random. The removed constraint is used for the assembly of the subsystems its removal generates. The method is illustrated in Fig. 17.

Properties

This algorithm is correct with respect to the WCM characterization of rigidity. It runs in polynomial time provided that the witness configuration is obtained in polynomial time: the number of anchors is polynomial, generally $O(n^2)$ in 2D and $O(n^3)$ in 3D, and each constraint is removed at most once. Note that a single witness configuration is used during the whole process.

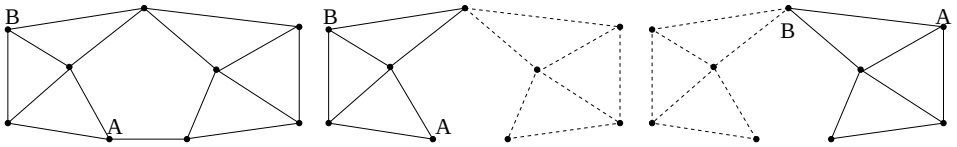

Fig. 17. First steps of the `WCM-RD` decomposition applied to a 2D bar framework. Maximal rigid parts with anchor AB are represented with solid lines. Middle and right: Removing one constraint creates two maximal rigid parts.

10. Conclusion

Decomposition methods are appealing for the drastic gain in efficiency they offer (see Ref. [22] for a performance comparison). Moreover, decomposition methods help the user to debug its constraint systems by identifying under-/over-constrained components before they are solved and by localizing, inside small subsystems, the problems occurring at solving time.

Significant progress has been accomplished during the last decade. The rigidity theory has brought a solid base to the geometric methods. Although the four categories of approaches have gained in generality and reliability, a major effort needs still be done to address the most challenging applications like CAD. We foresee that the design of novel hybrid approaches will allow decisive advances toward practical requirements. To be specific, we think that a method that resorts as much as possible to predefined guarded patterns, and uses a general DOF-based constriction check enhanced by a WCM-based validation in a recursive assembly fashion (allowing to interleave decomposition and recombination phases), would be far better than any existing method w.r.t. generality and reliability.

Acknowledgments

The authors would like to heartily thank Dominique Michelucci and Pascal Schreck for their involvement and invaluable advices in preparing this survey. Also thanks to Marc Gouttefarde for useful comments.

References

1. C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Decomposition of Geometric Constraints Part I: performance measures & Part II: new algorithms. *J. of Symbolic Computation*, 31(4), 2001.
2. C.M. Hoffmann and R. Joan-Arinyo. A brief on constraint solving. *Computer-Aided Design and Applications*, 2005.
3. M. Sitharam. Combinatorial approaches to geometric constraint solving: problems, progress, directions. In *AMS-DIMACS book on CAD and manufacturing*. 2005.
4. C. Jermann. *Résolution de contraintes géométriques par rigidification récursive et propagation d'intervalles*. Ph.d. thesis, UNSA, NICE, 2002.
5. L. Henneberg. *Die graphische Statik der starren Systeme.* , Leipzig, 1911.
6. I. Fudos. *Constraint Solving for Computer Aided Design*. PhD thesis, Purdue University, 1995.

7. J. Graver, B. Servatius, and H. Servatius. *Combinatorial Rigidity. Graduate Studies in Mathematics*. American Mathematical Society, 1993.
8. L. Lovasz and Y. Yemini. On generic rigidity in the plane. *SIAM J. Alg. Discrete Methods*, 3:91–98, 1982.
9. H. Crapo. On the generic rigidity of plane frameworks. Technical Report 1278, INRIA, 1990.
10. Bruce Hendrickson. Conditions for unique realizations. *SIAM journal of Computing*, 21(1):65–84, 1992.
11. W. Whiteley B. Servatius. Constraining Plane Configurations in Computer-Aided Design: Combinatorics of Directions and Lengths. *SIAM J. on Discrete Mathematics*, 12(1):136–153, 1999.
12. Y.Zhou M. Sitharam. A tractable, approximate characterization of combinatorial rigidity in 3D. In *ADG'2004*, 2004.
13. M. Sitharam. *Frontier, an opensource gnu geometric constraint solver : version3 (2003) for general 2d and 3d systems*, 2003. http://www.cise.ufl.edu/ sitharam.
14. C. Jermann, B. Neveu, and G. Trombettoni. A new structural rigidity for geometric constraint systems. In *Int. Workshop on Automated Deduction in Geometry, ADG'2002, LNCS 2930*, pages 87–106, 2004.
15. G.-F. Zhang X.-S. Gao. Geometric constraint solving based on connectivity of graph. Technical Report 22, Academia Sinica, Beijing, China, december 2003.
16. A. S. Householder. *Principles of Numerical Analysis*. McGraw-Hill, New York, NY, USA, 1953.
17. E. Lahaye. Une méthode de résolution d'une catégorie d'équations transcendantes. *Compte-rendu des Séances de L'Académie des Sciences*, 198:1840–1842, 1934.
18. D. Michelucci and H. Lamure. Résolution de contraintes géométriques par homotopie. In *Actes de AFIG 1994*, 1994.
19. C. Durand. *Symbolic and Numerical Techniques For Constraint Solving*. PhD thesis, Purdue University, 1998.
20. R.E. Moore. *Interval Analysis*. Prentice-Hall, 1966.
21. C. Jermann, G. Trombettoni, B. Neveu, and M. Rueher. A Constraint Programming Approach for Solving Rigid Geometric Systems. In *Constraint Programming, CP'2000*, volume 1894 of *LNCS*, pages 233–248, 2000.
22. C. Jermann, B. Neveu, and G. Trombettoni. Inter-Block Backtracking: Exploiting the Structure in Continuous CSPs. In *Selected papers of the Int. Works. on Global Optim. and Constraint Satisfaction, COCOS 2003, LNCS 3478*, pages 15–30, 2005.
23. W. Wu. Basic principles of mechanical theorem proving in elementary geometries. *J. Automated Reasoning*, 2:221–254, 1986.
24. S.C. Chou. *Mechanical Theorem Proving*. Reidel Publishing Co., 1988.
25. P. Mathis, P. Schreck, and J.-F. Dufourd. Yams : A multi-agent system for 2d constraint solving. In Beat Bruderlin and Dieter Roller, editors, *Geometric Constraint Solving and Applications*, pages 211–233. Springer, 1998.
26. M. Wilczkowiak, G. Trombettoni, C. Jermann, P. Sturm, and P. Boyer. Scene modeling based on constraint system decomposition techniques. In *9th IEEE International Conference on Computer Vision, ICCV'03*, 2003.
27. J. Owen. Algebraic solution for geometry from dimensional constraints. In *Proc. of Solid Modeling and CAD/CAM Applications*, pages 397–407, 1991.
28. J.-F. Dufourd, P. Mathis, and P. Schreck. Geometric construction by assembling solved subfigures. *Artificial Intelligence*, 99(1):73–119, 1998.
29. I. Fudos and C.M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, 16(2):179–216, 1997.
30. R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana-Pasto. Transforming an under-constrained geometric constraint problem into a well-constrained one. In *SM'03: Proc. of Solid modeling and applications*, 2003.

31. R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana-Pasto. Revisiting decomposition analysis of geometric constraint graphs. *Computer Aided Design*, 36:123–140, 2004.
32. J.E. Hopcroft and R.E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Computing*, 3:135–158, 1973.
33. C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Finding solvable subsets of constraint graphs. In *Proc. of Constraint Programming, CP'97*, pages 463–477, 1997.
34. A. Lomonosov. *Graph and combinatorial algorithms for geometric constraint solving.* PhD thesis, University of Florida, 2004.
35. J.J. Oung, M. Sitharam, B. Moro, and A. Arbree. Frontier: fully enabling geometric constraints for feature based modeling and assembly. In *Proceedings of ACM Solid Modeling symposium, 2001*, 2001.
36. C.M. Hoffmann, M. Sitharam, and B. Yuan. Making constraint solvers more usable: Overconstraint problem. *Computer Aided Design*, 36(4):377–399, 2004.
37. S.-M. Hu Y.-T. Li and J.-G. Sun. A Constructive Approach to Solving 3D Geometric Constraint Systems Using Dependence Analysis. *Computer Aided Design*, 34(2):97–108, 2002.
38. X.-S. Gao and G.-F. Zhang. Geometric Constraint Solving via C-tree Decomposition. In *SM'03*, pages 45–55, 2003.
39. G. Sunde. Specification of shapes by dimensions and other geometric constraints. In *IFIP WG 5.2 Geometric Modeling*, 1986.
40. A. Verroust, F. Schonek, and D. Roller. Rule oriented method for parametrized computer aided design. *Computer Aided Design*, 24(6):531–540, 1992.
41. R Joan-Arinyo and A Soto. A correct rule-based geometric constraint solver. *Computer and Graphics*, 5(21):599–609, 1997.
42. W. Bouma, I. Fudos, C.M. Hoffmann, J. Cai, and R. Paige. Geometric constraint solver. *Computer Aided Design*, 27(6):487–501, 1995.
43. I. Fudos and C.M. Hoffmann. Correctness proof of a geometric constraint solver. *International Journal of Computational Geometry and Applications*, 6:405–420, 1996.
44. C.M. Hoffmann and R. Joan-Arinyo. Symbolic constraints in constructive geometric constraint solving. *Journal of Symbolic Computation*, 23:287–299, 1997.
45. G. Kramer. *Solving Geometric Constraint Systems.* MIT Press, 1992.
46. M. Gruebler. *Getriebelehre.* Springer, Berlin, 1917.
47. X.S. Gao, K. Jiang, and C.-C. Zhu. Geometric constraint solving with conics and linkages. *Computer Aided Design*, 34(6):421–433, 2002.
48. P. Schreck. Implantation d'un système à base de connaissances pour les constructions géométriques. *Revue d'Intelligence Artificielle*, 8(3):223–247, 1994.
49. S.C. Chou, X.S Gao, and J. Z. Zhang. A deductive database approach to automated geometry theorem proving and discovering. *Journal of Automated Reasoning*, 25(3):219–246, 2000.
50. B. Aldefeld. Variations of geometries based on a geometric-reasoning method. *Computer-Aided Design*, 20(3):117–126, 1988.
51. P. Schreck. Robustness in CAD Geometric Construction. In *Proc. of the International Conference on Information Visualisation, IV'2001*, pages 111–116. IEEE, 2001.
52. E. Schramm and P. Schreck. Solving geometric constraints invariant modulo the similarity group. In *Proc. of the 2002 Int. Conference on Computational Science and its Applications, Part II*, pages 356–365. LNCS 2669 (Part III), 2003.
53. P. Schreck and P. Mathis. Geometrical constraint system decomposition: a multigroup approach. Technical report, Université de Strasbourg, France, 2005.
54. D. Serrano. *Constraint Management in Conceptual Design.* PhD thesis, MIT, 1987.
55. J.E. Hopcroft and R.M. Karp. An $n^{2.5}$ algorithm for maximum matching in bipartite graphs. *SIAM J. Computing*, 2(4):225–231, 1973.
56. R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Com-*

puting, 1(2):146–160, 1972.
57. D. Koenig. Über Graphen und ihre Anwendung auf Determinantentheorie und Mengenlehre. In *Math Ann 77*, pages 453–465, 1916.
58. A. Pothen and C.J. Fan. Computing the block triangular form of a sparse matrix. *ACM Trans. on Math. Soft.*, 16(4):303–324, 1990.
59. R.S. Latham and A.E. Middleditch. Connectivity analysis: A tool for processing geometric constraints. *Computer Aided Design*, 28(11):917–928, 1996.
60. I. Sutherland. *Sketchpad: A Man-Machine Graphical Communication System*. PhD thesis, Department of Electrical Engineering, MIT, 1963.
61. A. Borning. *ThingLab: A Constraint-Oriented Simulation Laboratory*. PhD thesis, Stanford University, 1979.
62. B. Vander Zanden. An incremental algorithm for satisfying hierarchies of multi-way, dataflow constraints. *ACM TOPLAS*, 18(1):30–72, 1996.
63. S. Ait Aoudia. *Modélisation géométrique par contraintes : quelques méthodes de résolution*. PhD thesis, Ecoles des Mines de Saint Etienne, 1994.
64. C. Hsu and B. Brüderlin. A degree-of-freedom graph approach. In *Geometric Modeling: Theory And Practice*, pages 132–155. , 1997.
65. P. Massan Kuzo. *Des contraintes projectives en modélisation tridimensionnelle interactive*. PhD thesis, Ecole des Mines de Nantes, 1999.
66. E. Eremchenko and A. Ershov. Two new decomposition techniques in geometric constraint solving. Research report Preprint number 11, LEDAS Company, 2004.
67. C. Bliek, B. Neveu, and G. Trombettoni. Using Graph Decomposition for Solving Continuous CSPs. In *Proc. of Constraint Programming, CP'98*, volume LNCS 1520, pages 102–116, 1998.
68. G. Trombettoni. A polynomial time local propagation algorithm for general dataflow constraint problems. In *Proc. of Constraint Programming*, volume LNCS 1520, pages 432–446, 1998.
69. G. Trombettoni and M. Wilczkowiak. GPDOF: a fast algorithm to decompose underconstrained geometric constraint systems: Application to 3D model reconstruction. *Int. Journal of Computational Geometry and Applications (IJCGA)*, 16, 2006.
70. P. Mathis. *Constructions géométriques sous contraintes en modélisation à base topologique*. PhD thesis, Université Louis Pasteur, 1997.
71. C. Essert-Villard, P. Schreck, and J.F. Dufourd. Sketch-based pruning of a solution space within a formal geo. constraint solver. *Arti. Intelligence*, 124:139–159, 2000.
72. S. Foufou, D. Michelucci, and J.-P. Jurzak. Numerical decomposition of geometric constraints. In *Proc. of Solid and Physical Modeling, SPM*, pages 143–151, 2005.
73. D. Michelucci and S. Foufou. The Witness Configuration Method. *Computer Aided Design*, 2005.
74. W. Whiteley. Applications of the geometry of rigid structures. In Henry Crapo, editor, *Computer Aided Geometric Reasoning*, pages 219–254. INRIA, 1987.
75. H. Lamure and D. Michelucci. Qualitative study of geometric constraints. In *Geometric Constraint Solving and Applications*, pages 234–258. Springer, 1998.
76. J. Graver. *Counting on Frameworks: Mathematics to Aid the Design of Rigid Structures*. Number 25. Mathematical Association of America, 2002.
77. W.A. Martin. Determining the equivalence of algebraic expressions by hash coding. *J. ACM*, 18(4):549–558, 1971.