Algorithmes pour la détection de rigidité dans les systèmes de contraintes géométriques

Christophe Jermann
AI Lab. - EPFL, 1015 Lausanne, Switzerland
e-mail: Christophe.Jermann@epfl.ch

Bertrand Neveu   Gilles Trombettoni
COPRIN Team, INRIA-I3S/CNRS-CERMICS,
2004 route des lucioles, BP 93, 06902 Sophia Antipolis, France
e-mail: Bertrand.Neveu@sophia.inria.fr,
Gilles.Trombettoni@sophia.inria.fr

Abstract

The structural rigidity property, a generalization of Laman's theorem which characterizes rigid bar frameworks in 2D, is generally considered a good approximation of rigidity in geometric constraint satisfaction problems (GCSPs). However, it may fail even on simple GCSPs because it does not take geometric properties into account.

In this paper, we question the flow-based algorithm used by Hoffmann *et al.* to identify rigid subGCSPs. We show that this algorithm may fail because of the structural rigidity, but also by design. We introduce a new flow-based algorithm which uses Jermann *et al.*'s characterization of rigidity. We show that this algorithm is correct in 2D and 3D, and can be used to tackle the major issues related to rigidity: deciding whether a GCSP is rigid or not and identifying rigid (or over-rigid) subGCSPs.

1 Introduction

Geometric constraint satisfaction problems (GCSPs) arise naturally in several areas, such as CAD, robotics and molecular biology. The rigidity concept is in the heart of many of these problems: deciding whether a GCSP is rigid or not, detecting rigid or over-rigid sub-parts, and so on.

Several methods [**?**, **?**, **?**, **?**, **?**, **?**] for solving GCSPs have to handle rigidity related problems. In particular, geometric decompositions of GCSPs produce sequences of rigid subGCSPs to be solved separately and then assembled.

The techniques used so far for rigidity detection can be classified in two categories: pattern-based approaches [**?**, **?**] depend on a repertoire of rigid bodies of known *shape* which cannot cover all practical instances; flow-based approaches [**?**, **?**] use flow (or maximum matching) machinery to identify subGCSPs verifying a structural property: the *structural rigidity*. This property is based on a degree of freedom count.

The latter approaches are more general eventhough structural rigidity is only an approximation of rigidity. Heuristics, like *ad-hoc* geometric rules, have been proposed to enhance structural rigidity capabilities, none of which succeeded to cover the gap between structural rigidity and rigidity. [**?**] have defined the *extended structural rigidity*, a new approximation of rigidity which supersedes even the heuristically enhanced classical characterizations.

In this paper, we focus on the algorithmic aspects of rigidity structural characterization. [**?**] have proposed a flow-based algorithm called `Dense` for this purpose. After providing the necessary background (Section 2), we exemplify the limits of this algorithm and the capabilities of our new algorithm (Section 3). Section 4 presents the specificities of our new algorithm and explains its advantages: first, it uses the extended structural rigidity instead of the structural rigidity; second it uses flow machinery in a geometrically correct way. We explain how this algorithm can be used to tackle the major issues related to the rigidity concept.

2 Background

This section provides the necessary background for the paper. It formally defines GCSPs, the rigidity concept and the structural characterizations of rigidity.

2.1 Geometric Constraint Satisfaction Problems

Definition 1 GCSP
A **geometric constraint satisfaction problem** *(GCSP) $S = (O, C)$ is defined by a set of geometric objects O and a set of geometric constraints C binding its objects.*
$S' = (O', C')$ is a **subGCSP** *of $S = (O, C)$ (noted $S' \subset S$) iff $O' \subset O$ and every constraint in $C' \subset C$ binds only objects in O' (i.e., S' is induced by O').*

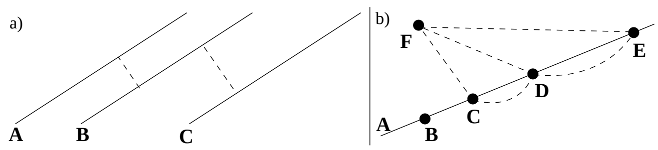

Figure 1: Two examples of GCSPs

Fig. 1-a presents a GCSP in 2D composed of 3 lines constrained by 2 parallelisms and 2 line-line distances; Fig. 1-b depicts a GCSP in 3D composed of 1 line and 5 points bound by 4 point-line incidences and 5 point-point distances.

We assume that geometric objects are indeformable (e.g., no circle with variable radius). Also geometric constraints must involve only positions and orientations of the objects and they must be independent from the global reference system (i.e., constraints only fix objects relatively one to another). These limitations make the structural characterizations of rigidity easier and are mandatory for geometric solving methods based on rigidity.

According to these restrictions, a solution to a GCSP $S = (O, C)$ is composed of one position and orientation for each object in O and satisfies all the constraints in C. For the solving purpose, a GCSP is translated into a system of equations: each object is represented by a set of unknowns (over the reals) which determine its position and orientation; each constraint becomes a system of equations on the unknowns of the objects it constrains.

2.2 Rigidity

Rigidity is defined w.r.t. *movements*. A movement in a GCSP is either a *deformation* if the relative positions of the objects are not preserved, or a *displacement* (rotation+translation). Intuitively, a GCSP is *rigid* if it admits no deformation, and all the displacements of the geometric space. It is *under-rigid* if it admits some deformations, and *over-rigid* if it does not admit some displacements or has no solution. More formal definitions of rigidity can be found in [**?**].

In the example of figure 1-b, the subGCSP CDF is rigid since a triangle is indeformable and can be displaced anywhere in the Euclidean 3D space. The subGCSP AF is under-rigid: point F can move independently of line A since there is no constraint between them. The subGCSP $ACDEF$ is over-rigid since it has no solution: generically, it is impossible to place point F at the intersection of the 3 spheres (distances) with aligned centers C, D and E.

2.3 Structural Rigidity

The *structural rigidity* corresponds to an analysis of *degrees of freedom* (DOF) in a GCSP. Intuitively, one DOF represents one independent movement in a GCSP. More formally:

Definition 2 Degree of freedom (DOF)
- Object: DOF(o) is the number of independent parameters used to determine the position and the orientation of o.
- Constraint: DOF(c) is the number of independent equations in the subsystem of equations representing c.
- GCSP: DOF(S)=$\sum_O$DOF(o) - $\sum_C$DOF(c) if $S = (O, C)$.

In 3D, points have 3 DOFs, lines have 4 DOFs; point-line incidences remove 2 DOFs, and point-point distances remove 1 DOF. Thus, subGCSPs ACD, CDF and AF from the GCSP in Fig. 1-b have respectively 5, 6 and 7 DOFs.

Structural rigidity is a generalization of Laman's theorem [**?**], which characterizes generic rigidity of 2D bar frameworks through a DOF analysis, to GCSPs composed of objects and constraints of any type and in any dimension. In practice, it is considered a *good approximation* of rigidity [**?**, **?**, **?**].

Definition 3 Structural rigidity (s_rigidity)
A GCSP $S = (O, C)$ *in dimension* d *is* **s_rigid** *iff DOF(S) =* $\frac{d(d+1)}{2}$ *and* $\forall S' \subset S$, *DOF(*S'*)* $\geq \frac{d(d+1)}{2}$.
S *is* under-s_rigid *iff DOF(S) >* $\frac{d(d+1)}{2}$ *and contains no over-s_rigid subGCSP.*
S *is* over-s_rigid *iff* $\exists S' \subseteq S$, *DOF(*S'*) <* $\frac{d(d+1)}{2}$.

The gap between rigidity and s_rigidity is in fact significant (see [**?**]). We illustrate the difference on two examples borrowed from Fig. 1-b: the subGCSP $ABCD$ is s_rigid since it has DOF($ABCD$)=6 in 3D, but it is in fact under-rigid since point B can move independently of segment CD along line A; the subGCSP $ACDE$ is over-s_rigid since it has only 5 DOFs, but it is in fact well-rigid.

2.4 Extended Structural Rigidity

The extended structural rigidity (es_rigidity in short) is based on the *degree of rigidity* (DOR) concept. Intuitively, the degree of rigidity of a subGCSP represents the number of independent displacements it admits with respect to the geometric properties between its objects. For example, a subGCSP S' composed of two lines in 2D has DOR(S')=3 if the two lines are not parallel, but DOR(S')=2 if the lines are parallel; the parallelism of the lines can be an explicit constraint, but it can also be induced by the constraints of the GCSP S' belongs to. In this second case, computing the DOR may be equivalent to geometric theorem proving.

The principle behind the extended structural rigidity is the following: a GCSP is rigid if all its movements are displacements; since the number of DOFs represents the number of independent movements while the DOR represents the number of independent displacements, comparing both allows us to determine if a GCSP admits some movements which are not displacements, i.e., deformations.

Definition 4 Extended Structural Rigidity (es_rigidity)
A GCSP $S = (O, C)$ *in dimension* d *is* **es_rigid** *iff DOF(S)=DOR(S) and* $\forall S' \subset S$, *DOF(*S'*)≥DOR(*S'*).*
S *is* under-es_rigid *iff DOF(S) > DOR(S) and contains no over-es_rigid subGCSP.*
S *is* over-es_rigid *iff* $\exists S' \subseteq S$, *DOF(*S'*)<DOR(*S'*).*

The es_rigidity is superior to the s_rigidity (e.g., es_rigidity exactly corresponds to rigidity on the GCSPs in Fig. 1). See [**?**] for a comparison between s_rigidity and es_rigidity and details about the DOR concept.

2.5 Object-Constraint Network

A GCSP S is transformed into an *object-constraint network* $G = (s, V, t, E, w)$. Fig. 2-a depicts the object-constraint network of the GCSP presented in Fig. 1-a.

This network was introduced in [**?**] for the algorithm `Dense`. A flow in this network represents a distribution of the DOFs of the constraints onto the DOFs of the objects. The principle of structural rigidity detection is to apply an overflow K in the network in order to identify subGCSPs having less than K DOFs left. In Section 4, we will explain how this principle is extended to search for s_rigid (Hoffmann *et al.*'s `Dense` algorithm) and es_rigid (our new algorithm) subGCSPs.

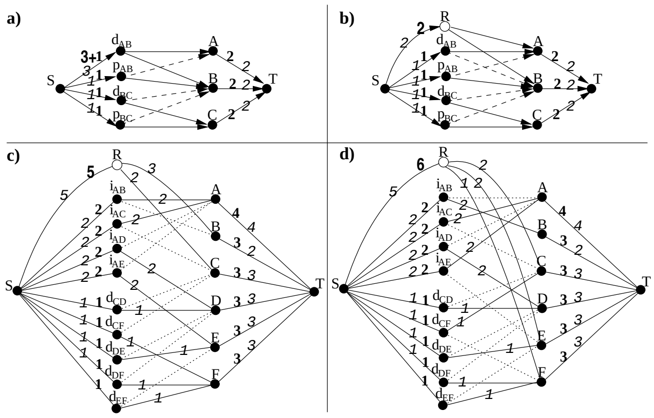

Figure 2: Object-constraint networks and flow distributions

Definition 5 Object-Constraint Network (s, V, t, E, w)
- s is the source and t is the sink.
- Each object $o \in O$ becomes an object-node $v_o \in V$.
- Each constraint $c \in C$ becomes a constraint-node $v_c \in V$.
- For each object $o \in O$ there is an arc $v_o \rightarrow T$ of capacity $w(v_o \rightarrow T)$ =DOF(o) in E.
- For each constraint $c \in C$, there is an arc $S \rightarrow v_c$ of capacity $w(S \rightarrow v_c)$ =DOF(c) in E.
- For each object $o \in O$ constrained by $c \in C$ there is an arc $v_c \rightarrow v_o$ of capacity $w(v_c \rightarrow v_o) = \infty$ in E.

3 Overview

In this section, we exemplify the contribution presented in this paper on the two GCSPs displayed in Fig. 1. These examples illustrate the two main differences between algorithm `Dense` and our algorithm:

1. In our algorithm, the value of the overload depends on the geometric properties of the objects it is applied to, while it is a constant[1] in `Dense`.

2. the overload is applied via a dedicated node R which can be attached to any set of objects in our algorithm, while it is applied directly via one constraint-node in `Dense`.

[1]This constant depends on the dimension of the geometric space since it represents the number of independent displacements (rotation+translation): 3 in 2D, 6 in 3D.

3.0.1 Example 1

The first example (Fig. 1-a; in 2D) highlights the first difference. Fig. 2-a presents the object-constraint network associated to this GCSP. In this picture, one can see the overload $K = 3$ applied on the first constraint by algorithm `Dense`. This constraint is linked to two lines, A and B, which are parallel and lie at prescribed distance in the plane; AB is a rigid subGCSP. However, one can easily see that the overload cannot be distributed completely since a capacity 5 (two constraints plus the overload) is applied to two lines having only 4 DOFs. Hence, the GCSP is identified as over-rigid since it contains a sub-GCSP with less than 3 DOFs.

Fig. 2-b displays our algorithm behavior when the virtual constraint R is linked to the same subGCSP, AB. The value of the overflow K is computed according to the geometric properties of these lines: since they are parallel, $K = 2$ (instead of 3 in algorithm `Dense`). Thus, the flow can be saturated: a capacity 4 (two constraints plus the overflow) exactly matches the 4 DOFs of AB; the GCSP is not identified over-rigid by our algorithm. Further overflow applications would allow to identify the GCSP as well-rigid.

3.0.2 Example 2

The second example (Fig. 1-b; in 3D) illustrates the second difference. Its object-constraint network is depicted in Fig. 2-d. This figure shows the application of an overflow 6 via the virtual constraint R onto the 3 points C, E and F by our algorithm; The overflow cannot be distributed completely, which signals an over-rigid subGCSP: $ACDEF$, found by adding reachable objects from R in the residual graph.

Algorithm `Dense` applies the overflow directly through a constraint-node. Since all constraints are binary in this example, `Dense` cannot apply an overflow to the same set of objects as our algorithm. More generally, `Dense` cannot apply the overflow to all subGCSPs and can miss rigid or over-rigid ones. Moreover, applying the overflow 6 to a pair of objects in this GCSP leads to an incorrect answer, as it was the case in the previous example; e.g., segments which are rigid would be identified over-rigid.

These examples show that some simple and very common subGCSPs in 2D and 3D, like parallel lines, triangles or segments, cannot be treated correctly by algorithm `Dense`[2].

In the following section, we detail the differences between algorithm `Dense` and our new algorithm and we present their consequences.

4 Algorithms

In this section, we present Hoffmann *et al.*'s `Dense` algorithm in comparison to our new algorithm. Both use flow machinery on the object-constraint network representing the GCSP. Our algorithm has two main differences with algorithm `Dense`:

[2] In practice [**?**], `Dense` embeds heuristic rules to prevent this kind of simple failures, but more complicated examples can still mistake the algorithm since no rule-based approach can handle all the singular cases.

- It uses es_rigidity instead of s_rigidity.
- It distributes flow in a *geometrically correct* way in the network.

These new features are achieved thanks to two major modifications in the `Distribute` function used by `Dense` (see beginning of Section 3).

We introduce first the principle of flow-based structural identification of rigidity; then we present and discuss function `Distribute` which is the key to our modifications. Finally we explain how function `Distribute` is used to design algorithms for the main problems related to rigidity.

4.1 Flow-based Rigidity Detection

From the geometric point of view, the principle of structural characterization of rigidity is to check if a GCSP admits only displacements. Hence, flow-based rigidity identification can be understood as follows:

1. remove K displacements by introducing K DOFs on the constraint side;
2. check if an over-constrained subGCSP S' exists by computing a maximum flow;
3. if so S' verifies DOF(S')<K.

Indeed, a maximum flow in the object-constraint network represents an *optimal* distribution of the DOFs of the constraints among the DOFs of the objects. If it does not saturate all the arcs outgoing from the source, some constraints' DOFs cannot be absorbed by the objects, i.e., the GCSP is over-constrained. In this case, there exists a subGCSP S' such that DOF(S')<0.

Thus, when an overflow equal to K is applied in the network on the constraint side, the identified subGCSP S' verifies DOF(S')<K. Hoffmann *et al.* have proven that S' is then induced by the objects traversed during the last search for an augmenting path during the maximum flow computation; in other words, S' is induced by the objects reachable from the overloaded constraint in the residual graph.

Depending on the value assigned to K, this principle can be applied to identify s_rigid ($K = \frac{d(d+1)}{2} + 1$), over-s_rigid ($K = \frac{d(d+1)}{2}$), es_rigid ($K = DOR + 1$) or over-es_rigid ($K = DOR$) subGCSPs.

4.2 Function `Distribute`

Function `Distribute` implements the principle presented above. It has been proposed in [**?**] as the core step of algorithm `Dense`. We present our version of this function and explain how it differs from Hoffmann *et al.*'s one.

As already said, applying an overflow K corresponds, from the geometric point of view, to removing K displacements from the objects linked to this constraint. But nothing ensures that the subGCSP linked to a single constraint allows K independent displacements: removing K DOFs from a subGCSP S' with DOR(S')<K is *geometrically incorrect*[3].

[3]Remember that the DOR represents the number of independent displacements of a subGCSP with respect to the geometric properties between its objects.

For instance, consider a subGCSP composed of 2 points linked by a point-point distance in 3D. This GCSP allows only 5 of the 6 (3 rotations + 3 translations) independent displacements of the 3D space since they lack the rotation around the line going through them. Therefore, removing 6 displacements from a couple of points is geometrically incorrect. However, Hoffmann *et al.*'s function `Distribute` does so when the distance constraint binding the two points in 3D is overloaded with $K = 6$.

In order to distribute the flow in a geometrically correct way, we propose to introduce a fictive constraint R, having DOF(R)=K. This constraint can be linked only to subset of objects O' allowing K independent displacements, i.e. inducing a subGCSP S' having DOR(S')$\geq K$. K and S' are two parameters of our function `Distribute`.

Distribute (S: GCSP; K: integer; S': GCSP) **returns** S'': GCSP)
Require: $K > 0$, $S' \subset S$ verifies DOR(S')$\geq K$
Ensure: $S'' \subset S$ verifies DOF(S'')$< K$, or S'' is empty
  $G \leftarrow$ `Overloaded-Network`(S, K, S')
  $V \leftarrow$ `FordFulkerson`(G)
  $S'' \leftarrow$ `Object-Induced-subGCSP`(V, S)
  Return S''

Function `Overloaded-Network` returns the object-constraint network corresponding to S where the fictive constraint R, set with capacity K, is linked to the objects of S'. The maximum flow computation is achieved by a standard flow algorithm[4] like `FordFulkerson` [**?**]. This function returns the set V of objects reachable from the virtual constraint R in the residual graph if the maximum flow cannot distribute the whole overload, an empty set V otherwise. Function `Object-Induced-subGCSP` returns the subGCSP S'' induced by V which verifies DOF(S'')<K or S'' is empty.

The two differences between our version of the `Distribute` function and Hoffmann *et al.*'s version have already been mentioned: the use of a dedicated constraint for overflow distribution, which allows to distribute the overflow to any subset of objects; and the adaptation of the overflow to the set of objects on which it is applied, which renders overflow application geometrically correct.

Example: Fig. 2-a presents the call to `Distribute(`$S, 3, dAB$`)` in Hoffmann *et al.*'s version for the GCSP in Fig. 1-a. Since the overflow cannot be fully distributed, the subGCSP AB is returned. This is correct from the flow point of view since DOF(AB)=2 is less than $K = 3$. However, from the geometric point of view, it is incorrect to interpret this result as an over-rigidity of the GCSP.

For the same subGCSP, our `Distribute` function leads to a different call and result: since DOR(AB)=2, the overflow can be at most 2. Fig. 2-b presents the call to `Distribute(`$S, 2, AB$`)`. The overflow can be distributed fully: no subGCSP is returned. Further similar calls would allow to conclude that this GCSP is not over-rigid.

Time Complexity: The complexity of our function `Distribute` is dominated by that of function `FordFulkerson`; it is $O(n^2(n+m))$ where n is the number of nodes

[4]In [**?**], function `Distribute` is specifically designed for binary constraints and flow distribution is merged with network construction and subGCSP identification.

and m the number of arcs. It is strictly equivalent to the complexity of Hoffmann *et al.*'s version.

Note that if several calls to this function are performed, it could be modified to compute maximum flow in an incremental way, yielding a better complexity.

4.3 Algorithms For Rigidity Detection

Based on the `Distribute` function, several algorithms can be designed to tackle the major problems related to the rigidity concept. Hoffmann *et al.* have proposed the `Dense` and `Minimal_Dense` algorithms to identify a well or over-rigid subGCSP and minimize it (using classical combinatorial minimization process, as it is done by `Minimal_Dense`). These algorithms can be reproduced using our function `Distribute`. This allows us to tackle the same problems in a geometrically correct manner and with a better characterization of rigidity: the extended structural rigidity. We will show on algorithm `Dense` how to introduce our `Distribute` function in existing algorithms.

Algorithm `Dense` Versus Algorithm `Over-Rigid`

Schematically, algorithm `Dense` operates by calling the `Distribute` function for each constraint in the GCSP until a non-empty subGCSP is returned. The overload is induced by the dimension of the considered geometric space: it represents the maximum number of independent displacements in this space (3 in 2D, 6 in 3D). `Dense` is supposed to return only over-rigid subGCSPs since returned GCSPs do not admit all the displacements allowed by the considered geometric space.

In fact, `Dense` is incorrect since it may remove more DOFs than the number of displacements admitted by a subGCSP. For instance, two parallel lines admit only 2 displacements in 2D; hence, removing 3 displacements from two parallel lines is geometrically incorrect in 2D.

To obtain a geometrically correct version of algorithm `Dense`, we propose to use the es_rigidity instead of the s_rigidity, i.e. the DOR is the overload; also, we use our `Distribute` function instead of Hoffmann *et al.*'s one.

This results in a new algorithm, called `Over-Rigid`, for structurally identifying over-rigid subGCSPs in a correct manner: for each $S' \subset S$, one call to function `Distribute(`S`,DOR(`S'`),`S'`)` can identify over-es_rigid subGCSPs. Indeed, if the call for a given S' returns a non-empty subGCSP S'', then it verifies DOF(S'') < DOR(S') a sufficient condition for being over-es_rigid (see Def. 4).

Over-Rigid (S: GCSP) returns S'': GCSP
Ensure: $S'' \subset S$ is over-es_rigid or empty
$S'' \leftarrow EmptyGCSP$
$M \leftarrow$ `DOR-Minimals(`S`)` *{builds the set M of all DOR-minimal subGCSPs in S}*
while $S'' = EmptyGCSP$ and $M \neq \emptyset$ **do**
$S' \leftarrow$ `Pop(`M`)`

$S'' \leftarrow$ `Distribute(`S`,DOR(`S'`),`S'`)`
end while
Return S''

Definition 6 DOR-minimal subGCSP
A subGCSP S' in a GCSP S is **DOR-minimal** *if it contains no proper subGCSP with the same DOR, i.e. $\forall S'' \subset S'$, DOR(S'')<DOR(S').*

4.4 Example of `Over-Rigid` application

Consider again the GCSP S presented in figure 1-b. Let $M = \{BC, BDF, CEF, ...\}$ be the set of DOR-minimal subGCSPs generated by `DOR-Minimals(`S`)`. Algorithm `Over-Rigid` then proceeds as follows:

1. First turn, $S' = BC$ and K =DOR(BC)=5. Fig. 2(c) represents the call to `Distribute(`$S, 5, BC$`)`. All the arcs outgoing from the source being saturated, no over-es_rigid subGCSP is identified.
2. At this turn, $S' = CEF$ and K =DOR(CEF)=6. Figure 2(d) represents the call to `Distribute(`$S, 6, CEF$`)`. This turn, the arc $S \rightarrow R$ is unsaturated. Since the set of object-nodes traversed during the last search for an augmenting path is $\{A, C, D, E, F\}$, the identified subGCSP is $ACDEF$ which is over-es_rigid.

On the same example, algorithm `Dense` would identify each segment (2 points + 1 distance) and each point on the line A as an over-rigid subGCSP, which is false.

4.5 Properties of algorithm `Over-Rigid`

The following proposition ensures that algorithm `Over-Rigid` is correct and complete.

Proposition 1 *Let S'' be the result of* `Over-Rigid`*(S). Either S'' is over-es_rigid, or it is empty and S contains no over-es_rigidity.*

Correctness of `Over-Rigid`: The correctness is based on Lemma 1 which states that the DOR of nested subGCSPs grows with their size.

Lemma 1 *Let S be a GCSP and $S' \subset S'' \subset S$ two subGCSPs. Then DOR(S') $\leq$ DOR(S'').*

Each unit of DOR in a GCSP represents an independent displacement (translation or rotation). Adding a new object o with some constraints to a subGCSP S' cannot remove the independent displacements already granted to S' since constraints are independent from the global reference system. Thus, DOR(S') $\leq$ DOR($S' \cup \{o\}$).□

Let S'' be a *non-empty* subGCSP resulting from `Over-Rigid`(S). Assuming it has been returned by the call to `Distribute`($S, DOR(S'), S'$), S'' must verify $DOF(S'') < DOR(S')$ since S'' is not empty. Moreover, by design of function `Distribute`, $S' \subset S''$. Since Lemma 1 implies that DOR(S')$\leq$DOR(S''), we can ensure that if S'' is not empty, then DOF(S'')<DOR(S''), i.e., S'' is over-es_rigid.□

Completeness of `Over-Rigid`: The completeness is based on Lemma 2 which states that overflow applications on nested subGCSPs with the same DOR are unnecessary.

Lemma 2 *Let $S'' \subset S' \subset S$ be two nested subGCSPs in a GCSP S. If the call to* `Distribute(`S, K, S'`)` *returns a* non-empty *subGCSP S_0, then the subGCSP S_1 returned by the call to* `Distribute(`S, K, S''`)` *is also* non-empty.

Let $G_{S'}$ be the object-constraint network overloaded for S' and $G_{S''}$ the network overloaded for S''. The only difference between these two networks resides in the fact that there are more arcs of the type $R \to o$ in $G_{S'}$. Thus, it is more difficult to distribute an overflow in $G_{S''}$ than in $G_{S'}$: if a maximum flow in $G_{S'}$ cannot saturate all the arcs outgoing from the source, a maximum flow in $G_{S''}$ cannot either.$\square$

Algorithm `Over-Rigid` applies an overload only for each element in the set M of all *DOR-minimal* subGCSPs (computed by `DOR-Minimals(`S`)`). Lemma 2 ensures that it is sufficient to distribute an overload for each DOR-minimal subGCSP, since any non DOR-minimal subGCSP contains, by definition, DOR-minimal subGCSPs.$\square$

Time Complexity of `Over-Rigid`: The complexity of algorithm `Over-Rigid` depends on the number of DOR-minimal subGCSPs. We have proven by enumeration that the number of objects in a DOR-minimal subGCSP is 2 in 2D and 3 in 3D for GCSPs including points, lines and planes constrained by distances, angles, incidences and parallelisms. Thus, for GCSPs in this class, the number of dor-minimal subGCSPs is $O(n^d)$ where n is the number of objects and d the dimension of the geometric space (2 or 3).

Let us call C_{DOR} the complexity of function `DOR-Minimals`, and C_{Dis} that of function `Distribute`, discussed in the previous section. Then, the worst-case complexity of algorithm `Over-Rigid` is $O(n^d * (C_{DOR} + C_{Dis}))$.

The complexity C_{DOR} is generally that of geometric theorem proving, i.e., it is exponential. However, in some practical classes of GCSPs, like mechanisms or bar frameworks, it is polynomial or even constant. Moreover, heuristic DOR computation can be used when geometric theorem proving is required but not affordable. In these cases, C_{DOR} can be neglected in comparison to C_{Dis}. We end up with $O(n^{d+2} * (n + m))$. In comparison, the complexity of algorithm `Dense` is $O(m * n^2 * (n + m))$. Thus, the overhead to obtain a geometrically correct algorithm is approximately linear in 2D, and quadratic in 3D.

4.6 Other algorithms

Function `Distribute` can be used in a similar way to tackle the major problems related to rigidity: identifying rigid subGCSPs (just by changing the value of the overflow in algorithm `Over-Rigid`), deciding if a GCSP is rigid (by one call to Over-Rigid and a DOF count), finding a minimal well- or over-rigid subGCSP (by classical minimization step, as in `Minimal_Dense`). For all these problems, using our new algorithms and the es_rigidity instead of the s_rigidity leads to geometrically correct and more reliable algorithms.

4.7 Applications

The new design of function `Distribute` allows a more general use of this function, in a geometrically sound manner: the overflow is not linked to the existing constraints but to a dedicated constraint; and the overflow should be distributed to any subset of objects instead of only to objects linked to a single constraint.

The DOR-minimal concept and its properties are the key to obtain a new family of polynomial algorithms for the major problems related to the rigidity concept.

These new algorithms can handle GCSPs in 2D and 3D correctly with respect to the es_rigidity. They can handle GCSPs with constraints like parallelism or incidence, which was not possible with Hoffmann *et al.*'s algorithms. Indeed, these constraints introduce geometric properties in GCSPs, leading to subGCSPs with DOR different from the maximum number of independent displacements of the considered geometric space. This kind of problematic constraints are ubiquitous in practical applications (architecture, CAD, mechanisms) and our new algorithms open a way for reliable industrial use.