A New Heuristic to Identify Rigid Clusters

Christophe Jermann*[1], Gilles Trombettoni[1], Bertrand Neveu[2], and Michel Rueher[1]

[1] {jermann, rueher, trombe}@essi.fr
Université de Nice–Sophia Antipolis, I3S, ESSI, 930 route des Colles, B.P. 145, 06903 Sophia Antipolis Cedex, France
[2] neveu@sophia.inria.fr
CERMICS, 2004 route des lucioles, 06902 Sophia Antipolis cedex, B.P. 93, France

Abstract. This paper presents a new heuristic for the identification of a rigid cluster in the recursive rigidification method. The heuristic we propose allows one to share objects between clusters in order to enhance the capabilities of the extension step during the planning phase in the recursive rigidification method. We introduce a new algorithm based on this heuristic and we prove the correctness of this algorithm.

1 Problem Description

The problem considered in this paper is the computation of all possible solutions of a geometric constraint system S.

Definition 1 *A geometric constraint system S is composed of*

- *A work space of given dimension d with a global reference system; for instance the Euclidean plane in dimension $d = 2$, with the Cartesian coordinate system.*
- *A set of geometric objects, such as points, lines, spheres, ... Each geometric object is modeled by a set of* generalized coordinates. *The generalized coordinates of a geometric object define its position and orientation in the work space.*
- *A set of geometric constraints on these objects, such as distance, incidence, angle, ... Each geometric constraint is modeled by a set of non-linear algebraic equations having for variables the generalized coordinates of the objects it involves.*

A solution for a geometric constraint system S gives a position and an orientation to each geometric object of S such that all the constraints of S are satisfied. In other words, it provides values for the generalized coordinates of every objects such that the system of algebraic equations made corresponding to all the constraints of S is consistent.

For the sake of simplicity, we will only consider geometric constraint systems involving the following objects of the Euclidean plane: points, lines and circles.

* Supported by CNRS and region Provence Alpes Côte d'Azur

However, the proposed approach can be extended to any objects in spaces of any dimension, as long as the following conditions hold:

- Geometric constraints involve only generalized coordinates as variables; i.e., geometric objects are non-deformable; for instance, no circle with variable radius.
- Geometric constraints are binary, i.e. involve two objects at most. However, the algebraic form of the constraints can involve more than two generalized coordinates.

The solving of such a system of geometric constraints is based on recursive rigidification. Rigidification techniques are standard graph-constructive methods taking advantage of the degrees of freedom of geometric objects [VSR92], [BFH+95], [FH93], [FH97], [DMS98], [Lee98].

In section 2 we present the recursive rigidification method and define all the notions employed by this method. Section 3 details the planning phase presented by Hoffmann et al. in [HLS97,HLS98]. The new heuristic we propose is presented in section 4. It is evaluated on an example. A new algorithm based on this heuristic is also presented and its correctness is established in section 5.

2 Recursive Rigidification

Recursive rigidification is a graph constructive method. It aims at decomposing the geometric constraint system S into rigid sub-systems to be solved one by one. It is composed of two main phases:

1. A **planning phase** which decomposes the geometric constraint system into rigid sub-systems. It also provides a partial order for the solving of these sub-systems.
2. A **solving phase** which solves all the sub-systems w.r.t. the partial order. Each sub-system is solved in a relative reference system. Finally all the sub-systems are assembled in a unique reference system using rigid-body transformations.

The planning phase is based on a degree of freedom analysis performed on a weighted geometric constraint graph $G = (O, C, w)$.

Definition 2 *A weighted geometric constraint graph $G = (O, C, w)$ is composed of:*

- *O the set of vertices. A vertex represent one geometric object of the corresponding geometric system.*
- *C the set of edges. An edge represent one geometric constraint of the corresponding geometric system.*
- *w a weight function.*
  For every vertex $o \in O, w(o)$ is the number of degrees of freedom of the geometric object corresponding to o; generally, it is the number of generalized

coordinates needed to fix the position and orientation of this object. For instance, a point in 2D has 2 (translational) degrees of freedom and then its corresponding vertex in G has weight 2
For all edge $c \in C, w(c)$ is the number of degrees of freedom fixed by the constraint corresponding to c; generally, it is the number of independent algebraic equations needed to model this constraint. For instance, a distance constraint fixes 1 degree of freedom and then its corresponding edge has weight 1.

The weight function w can also be applied to any sub-graph $G' = (O', C', w)$ of G. $w(G')$ computes the difference between the sum of vertices in O' weights and the sum of edges in C' weights: $w(G') = \sum o \in O' w(o) - \sum c \in C' w(c)$. This difference represents the degrees of freedom of the sub-system corresponding to the sub-graph G'.

The planning phase exploits a structural property of this graph to find rigid sub-graphs which can be solved one by one and then assembled. This structural property, called structural rigidity, or, in short, s-rigidity, is defined as follows:

Definition 3 *The weighted geometric constraint graph G corresponding to a geometric constraint system in dimension d is s-rigid iff:*

- $w(G) = \frac{d(d+1)}{2}$;
- *every sub-graph G' of G verifies:* $w(G') \geq \frac{d(d+1)}{2}$.

An s-rigid sub-graph of G is called a **cluster**.

The structural rigidity is similar to the property P introduced in [LM98] and to the *density* notion introduced in [HLS97]. The structural rigidity is a necessary condition to prove rigidity in the general case (*i.e.*, except in degenerate configurations [Hen92]). It is not a sufficient condition, except for distance constraints between points in 2D [Lam70]. However, if the system of corresponding equations is independent, the structural rigidity is a sufficient condition to ensure rigidity in the general case. In other cases, it is considered a good heuristic for detecting rigid sub-parts of a system.

The planning phase makes use of the s-rigidity property to produce a *plan*. A plan is a set of sub-graphs (equivalent to sub-systems) with a partial order between the sub-graphs. More precisely, the produced plan is a reversed tree of clusters, called cluster tree in the following.

The solving phase, also called construction phase, follows the plan given by the previous phase. It computes the coordinates of the geometric objects in every cluster. Then it recursively assembles clusters into a unique assembly.

3 The Planning Phase

The goal of the planning phase is to find an ordering for solving the constraints in an incremental way. More precisely, the aim is to identify rigid sub-parts, that can be solved independently and then assembled. Hoffmann et al. [HLS97] have introduced an algorithm which achieves such a planning. In this section, we

present and illustrate the algorithm proposed by Hoffmann et al. and we discuss its limits.

Hoffmann's planning algorithm builds a reverse tree of clusters called *cluster tree*: the root is the final cluster covering the whole system; the leaves are the clusters which involve no other cluster, but only geometric objects; there is an arc between a cluster K and all the clusters that have been merged to yield K.

The algorithm builds clusters in sequence by interleaving 3 steps until there is no more rigid sub-graph in the weighted geometric constraint graph. The 3 steps are the following ones:

- The **merge step** finds a cluster K of minimal size. This cluster is made of several prior clusters or geometric objects of G_m.
- The **extension step** repeats *single extensions* while the current cluster K remains s-rigid. One single extension adds to the current cluster K one object of G_m which is connected to K.
- The **updating step** removes from the graph all objects in the current cluster K and replaces them by a single vertex N_K of weight $\frac{d(d+1)}{2}$ in G_m. Edges between internal and external objects are transferred on this single vertex.

Let us illustrate this algorithm on an example in 2D made of 15 points and 27 distance constraints between them (see Figure 1 - G_m^0).

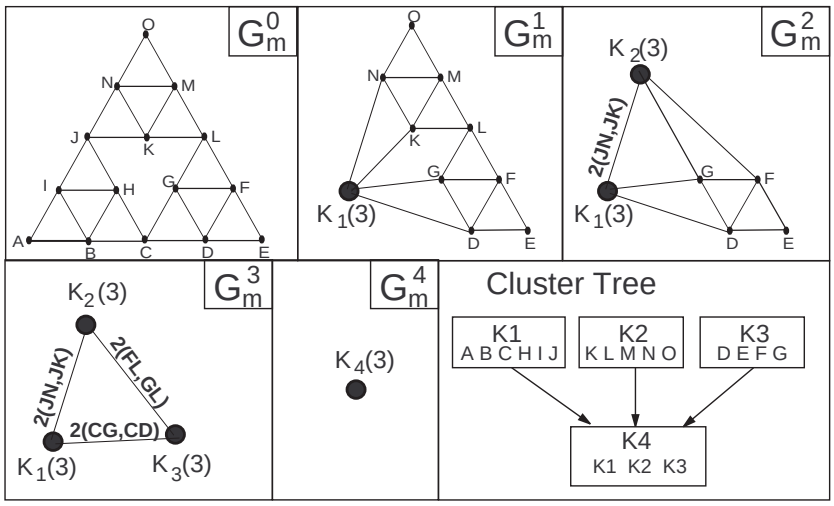

Fig. 1. Snapshots of the graph G_m during the execution of Hoffmann's planning method, and resulting cluster tree.

The first merge step finds the sub-graph $G =< A, B >$ of G_m. This cluster is extended until a fix-point is reached. The set of adjacent points of this cluster is $\{C, H, I\}$. Since $< A, B, I >$ is s-rigid, I is added to G. The same

process is performed to add H, C and J to G. So the new set of adjacent points is $\{D, G, N, K\}$. None of these points can be added by extension, so we have reached the fix-point $G =< A, B, I, H, C, J >$. A new vertex K_1 of weight 3 is added and A, B, I, H, C, J are removed from G_m as well as all constraints among them (see Figure 1 - G_m^1). K_1 is placed into the cluster tree and the next merge step is performed. It identifies $< M, N >$ as a minimal s-rigid sub-graph of G_m. This cluster is extended to $K_2 =< M, N, O, K, L >$ (see Figure 1 - G_m^2). Note that K_2 could have been extended onto J if $< M, N >$ were identified at the beginning. Finally, G, F, D and E are merged into K_3 (again, note that C and L could have been included in K_3) (see Figure 1 - G_m^3).
Since clusters do not share points, inter-cluster constraints (i.e., the distance constraints $dist(J, K)$, $dist(J, N)$, $dist(G, C)$, $dist(G, L)$, $dist(F, L)$ and $dist(C, D)$) are handled by the last merge step (see Figure 1 - G_m^4). So the solving process for this last step can be very expensive.

The critical observation is that one merge step, as well as one single extension onto one object, identifies one sub-system for the solving phase. The main difference between merge and extension resides in the fact that an extension can identify a solving sub-system of size at most $\frac{d(d+1)}{2}$ equations[1] while a merge step can identify a solving sub-system of any size.

Moreover, the merge step is the most expensive step of the planning phase. Indeed, the algorithm based on network flows techniques proposed by Hoffmann et al. in [HLS98] to perform one merge step on a weighted geometric constraint graph $G = (O, C, w)$ runs in $\mathbf{O}(\|O\|^2(\|O\| + \|C\|))$ while extension steps are linear operations.

Therefore, we propose a new heuristic where clusters can share objects, the aim being to maximize the extension capabilities, and thus, to reduce both the number of constraints which have to be handled during the merge steps and the number of required merge steps. This heuristic generalizes previous ad-hoc techniques [BFH+95].

4 The Proposed Heuristic

The goal of the heuristic we propose is to reduce the size of the identified sub-systems. We have shown on the example of previous section that the current cluster could have been extended to more objects if these objects were still present in the graph. The heuristic we propose is to leave these objects in the graph. Indeed, they are candidates to be included by extension in future clusters. A consequence of this heuristic is that clusters may share objects.

To apply this heuristic, we propose to use a new graph, G_e, dedicated to the extension step. Unlike G_m, G_e only contains geometric objects for vertices. G_e

[1] Since a non-deformable object in dimension d can have at most $\frac{d(d+1)}{2}$ degrees of freedom, and the position and orientation of any object having w degrees of freedom can be determined by solving a system involving w independent equations

is a more detailed model of the geometric constraint system than G_m is. Indeed, in G_e clusters are represented by the set of its *interface objects* and not by single vertex like in G_m. The *interface objects* are the objects of the cluster which are still connected to objects outside the cluster; the *internal objects* are the objects of the cluster which are connected only to other objects within the cluster. The interface objects of a cluster K are the candidates for the inclusion in future clusters, and thus they will remain in G_e. This imply that an object that has already been included in a cluster can be included in other clusters: clusters can share objects.

Let us describe the 3 steps of the new planning phase:

1. The merge step is still performed on G_m. The sub-graph K_m identified as a minimal cluster in G_m is then translated into the corresponding sub-graph K_e of G_e the following way: Each cluster in K_m is replaced by its set of interface objects; each object in K_m remains the same in K_e[2]. The translated system constitutes the solving sub-system identified by the merge step.
2. The extension step is performed like before, but in G_e instead of G_m: While K_e remains s-rigid, objects of G_e are added to K_e.
3. The updating step updates both G_e and G_m.
   In G_e, *interface constraints* are placed between the interface objects of K_e. These interface constraints are designed so that the sub-graph made of the interface objects along with these interface constraints is s-rigid.
   in G_m, K_m is replaced by a single vertex N_K, and *coincidence constraints* are placed between N_K and clusters with which K_m shares objects.

The algorithm `Rigidification` in the next section details the three steps of this new planning phase.

Shared Objects
Every object of the system is an interface object until it becomes an internal object. While an object remains an interface object, it is candidate to be shared by clusters . To handle the fact that one geometric object appears in different clusters, we duplicate this object: Each object o have one instance in each cluster in which it appears; the last cluster in which it is duplicated is the one where o becomes an internal object. However, vertices of G_e (and G_m) are not duplicated: one vertex represents one geometric object and all its instances.

Interface Constraints
Interface constraints are introduced to rigidify the set of interface objects of the newly formed cluster. We propose to use predefined *rigidification patterns* to place these interface constraints between the interface objects.

A rigidification pattern, called r-pattern in the following, is a set of geometric constraints which, applied to a particular set of objects O, rigidify these objects

[2] Performing the merge step on G_e is not possible since it could always select the same sub-graph: the sub-graph made of the interface objects and of the interface constraints of the first found cluster.

relatively to each other. For instance, a distance constraint is an r-pattern for 2 points or 1 point and 1 line in 2D.

Since it is impossible to define a rigidification pattern for any set of geometric objects, we propose to define only r-patterns for any set of 2 objects, and also to define r-patterns which rigidify any 1 object relatively to any 2 objects. These patterns allows us to triangulate the set of interface objects in the following way:

1. Rigidify any 2 interface objects o_1, o_2 of K using the corresponding r-pattern.
2. Rigidify any other interface object of K relatively to o_1 and o_2 using the corresponding rigidification pattern.

We propose the following set of r-patterns for points and lines in 2D:

- Rigidification of any 2 objects:
  - Points p_1 and p_2 are rigidified using a distance constraint $dist(p_1, p_2)$.
  - Point p_1 and Line l_1 are rigidified using a distance constraint $dist(p_1, l_1)$.
  - Lines l_1 and l_2 are rigidified using an angle constraint $angle(p_1, p_2)$.
- Rigidification of any 1 object relatively to any 2 other objects:
  - Point p relatively to:
    - Points p_1 and p_2: 2 distance constraints $dist(p, p_1)$ and $dist(p, p_2)$.
    - Point p_1 and Line l_1: 2 distance constraints $dist(p, p_1)$ and $dist(p, l_1)$.
    - Lines l_1 and l_2: 2 distance constraints $dist(p, l_1)$ and $dist(p, l_2)$.
  - Line l relatively to:
    - Points p_1 and p_2: 2 distance constraints $dist(l, p_1)$ and $dist(l, p_2)$.
    - Point p_1 and Line l_1: 1 distance constraint $dist(l, p_1)$ and 1 angle constraint $angle(l, l_1)$.
    - Lines l_1 and l_2: 1 angle constraint $angle(l, l_1)$ and 1 distance constraint between intersection points $l \cap l_1$ and $l \cap l_2$.

Using r-patterns is not sufficient to consistently rigidify a set of interface objects. Indeed, they must not only be rigid relatively to each other, but also respect the metric properties they have relatively to each other in every cluster where they occur. More precisely, for a given configuration c of the interface objects O of a cluster K_1, all produced configurations in a latter cluster K_2 which includes these objects O must be compatible with c, i.e. there must exist a rigid-body transformations such that the occurrences of O in K_1 and the occurrences of O in K_2 coincide.

To ensure this property, we propose:

1. To instantiate an interface constraint $c(A, B)$, placed by a cluster K_1 between its interface objects A and B, only when it is used, i.e. a single extension of another cluster K_2 utilizes this interface constraint $c(A, B)$ to include either A or B in K_2. The interface constraint is then instantiated between the instances of A and B in K_1 (A_1 and B_1) and the instances of A and B in K_2 (A_2 and B_2) the following way: $c(A_1, B_1) = c(A_2, B_2)$
2. To use orientation constraints to forbid unwanted configurations in our set of predefined r-patterns.

The use of interface constraints is illustrated in the following example.

Coincidence Constraints
A coincidence constraint is placed in G_m between any two clusters K_1 and K_2 sharing an object o of weight W. The weight placed on this coincidence constraint is W.
Note that since all sub-graphs of G_m are translated into sub-graphs of G_e, coincidence constraints are never involved in sub-systems to solve.

Example
We illustrate the behavior of the new heuristic on the small example presented in previous section (see Figure 2 - G_m^0 and G_e^0).

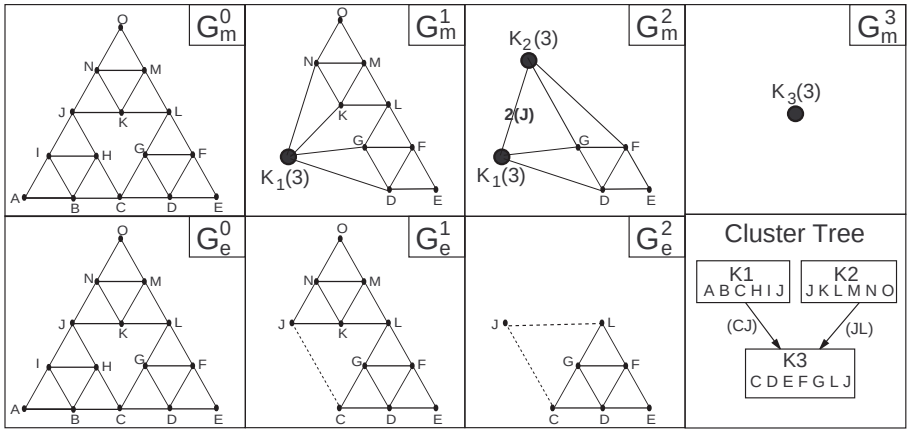

Fig. 2. Snapshots of the graphs G_m and G_e during the run of the algorithm, and obtained cluster tree. Interface constraints are drawn in dotted lines.

The first merge and extension steps yield a sub-graph G_1 containing the points A, B, I, H, C and J. The instances of these objects in K_1 will be called A_1, B_1, I_1, H_1, C_1 and J_1. A new vertex K_1 of weight 3 replaces these points in G_m (see Figure 2 - G_m^1). The internal points (A, B, H, I) and the internal constraints of K_1 are removed from G_e. An interface constraint is added between J and C (see Figure 2 - G_e^1). If we apply the r-patterns we have proposed, this interface constraint will be a distance constraint since J and C are points ($dist(C, J)$).

The cluster G_2 is then created in the same way. It is condensed into the vertex K_2 in G_m. G_2 contains instances J_2, K_2, L_2, M_2, N_2 and O_2 of the points J, K, L, M, N and O. Since G_2 includes the point J which also belongs to G_1, a coincidence constraint of weight $w(J) = 2$ is inserted in G_m between K_1 and K_2 (see Figure 2 - G_m^2). The interface objects of G_2 are J and L. A distance interface constraints is placed between these two points ($dist(J, L)$).

The cluster G_3 is finally created and condensed into K_3 in G_m. G_3 includes all the remaining vertices of G_e (see Figure 2 - G_m^3) and then contains instances

C_3, D_3, E_3, F_3, G_3, J_3, L_3. The interface constraints $dist(C, J)$ and $dist(J, L)$ placed by clusters K_1 and K_2 are used by the cluster K_3 to extends onto J. Thus, these constraints are instantiated between the occurrences of C, J and L in the different clusters:

- $dist(C, J)$ becomes $dist(C_1, J_1) = dist(C_3, J_3)$
- $dist(J, L)$ becomes $dist(J_2, L_2) = dist(J_3, L_3)$

This ensures that the metric properties between J_3, C_3 and L_3 solved in cluster K_3 will fit the metric properties of C_1, J_1, J_2, L_2 solved in the other clusters.

The planning phase is then finished and has produced the cluster tree depicted in Figure 2.

Evaluation

The execution of the planning algorithm proposed by Hoffmann et al. in [HLS97] on the same example yields a last merge step involving 6 constraints corresponding to a sub-system of at least 6 equations. Our heuristic allows us to limit the

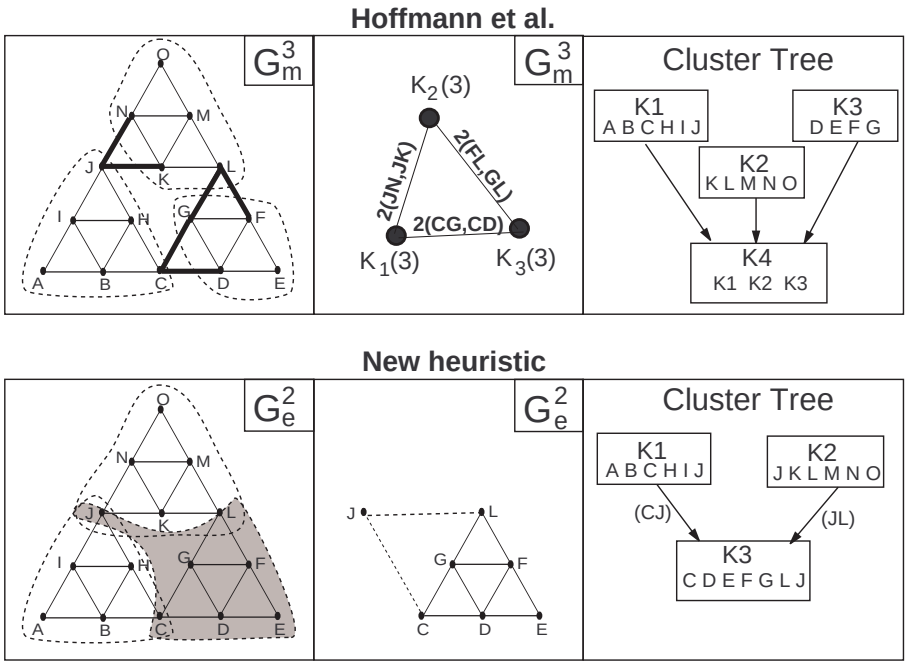

Fig. 3. Evaluation of our heuristic on the example

number of constraints involved in any identified sub-system to 2 constraints in the worst case (for this example). Moreover, using our heuristic, only 3 merge

steps were needed to fulfill the planning phase while 4 merge steps were needed without the heuristic.

Figure 3 illustrates these points. The left most graphs represent the 3 clusters obtained obtained without the heuristic (G_m^3) and the 3 other clusters obtained with the heuristic (G_e^2). Points J, L and C are shared in G_e^2.
The upper graph in the center represent the state of the weighted constraint graph after 3 merge steps without our heuristic (G_m^3); the graph below it represent the state of the weighted constraint graph after 2 merge steps using our heuristic (G_e^2). These two graphs are the last inputs given to the planning algorithm before it terminates.
The right most graphs represent the cluster trees obtained.

However, the sharing of objects implies to duplicate each object the number of times it is shared. This duplication adds more variables to the constraint system. Moreover, we introduce two kind of constraints (coincidence and interface constraints) which are also added to the initial system. Hence, the solving phase of a decomposition obtained using our heuristic involves a total of constraints and objects greater than the solving phase of a decomposition without our heuristic for the same initial system.

Despite this fact, experimentations show that it seems better to solve *more small blocks than less bigger blocks*. Comparisons on small examples are provided in [JTNR00]. These comparisons shows that using our heuristic, a decomposition in smaller or equal sub-systems is obtained: on our examples, we obtain sub-systems up to 6 times smaller. Also, the solving phase we propose in [JTNR00] is up to 6 times quicker for the decomposition of our examples with our heuristic than for the decomposition of the same systems without our heuristic.

5 The New Algorithm

This section present a planning algorithm called `Rigidification` which is based on our heuristic. We also prove the correctness of this algorithm and discuss its worst case complexity.

The function `MinimalSRigid`(G_m, d, G_e) which performs the merge step runs the flow-based algorithm presented by Hoffmann et al. in [HLS97].

Complexity Of `Rigidification`

The essential operation of the algorithm `Rigidification` is the merge step. Indeed, the extension step is only a linear operation, and the updating step can be considered a constant time operation.
The complexity of the `MinimalSRigid` function is given in [HLS97]: $\mathbf{O}(n^2(m+n))$ where n is the number of vertices in G and m is the number of edges in G.
The worst case complexity is reached when no extension is performed and each sub-graph identified by the `MinimalSRigid` function is reduced to a single edge. Then the complexity of the algorithm `Rigidification` is $\mathbf{O}(m * n^2(m+n))$.

Algorithm 1 `Rigidification` (**in** G: Graph; **in** d: Integer; **out** CT: Cluster-Tree)

{G is the initial weighted geometric graph; d is the dimension of the problem (2D, 3D); CT is the plan (cluster tree) that is produced by the algorithm.}
$CT \leftarrow \emptyset$; $G_m \leftarrow G$; $G_e \leftarrow G$
repeat
  $G_1 \leftarrow$ `MinimalSRigid`(G_m, d, G_e) *{Merge step}*
  {Extension step}
  repeat
    for all o of G_e connected by at least one edge to G_1 **do**
      if $G_1 \cup \{o\}$ is s-rigid **then**
        Add o and corresponding edges to G_1
      end if
    end for
  until G_1 is no more modified
  {Updating step}
  Let G_2 be the sub-graph of G_m corresponding to G_1
  Replace G_2 by a new vertex K in G_m
  Add in G_m the coincidence constraints for the shared objects of K
  Insert K in the cluster tree CT and add the dependencies for K
  Add in G_e interface constraints for the interface objects of G_1
  Remove from G_e internal objects of G_1 and connected arcs
until G_m contains no more rigid sub-graph

Correctness Proof Of `Rigidification`

The algorithm `Rigidification` terminates since the size of the graph G_m is reduced at each updating step. Indeed, after the processing of one sequence of merge, extension and updating steps, there is two different possible states:

1. Either a new non-empty cluster has been formed and then all objects and clusters it embeds are removed from G_m and replaced by a single vertex[3].
2. or no cluster has been found, i.e., the merge step has returned an empty sub-graph. This is the termination condition for the algorithm.

To prove the correctness of the algorithm `Rigidification`, we should prove the correctness of all the operations it performs. Proofs of merge and extension operations are provided by Hoffmann et al. in [HLS97]. So we should only prove the correctness of the updating step of the algorithm. To do so, we have to establish that each operation done while updating each graph preserves both the s-rigidity of the initial system and all the solutions of the initial system.

Thus, proving the following four propositions is sufficient to prove the correctness of our algorithm:

[3] Note that a non-empty cluster contains at least one edge, and then two vertices; hence, replacing the content of the cluster by a single vertex reduces the number of vertices in G_m by at least 1

Proposition 1. *G_m (resp. G_e) have the same weight before and after updating*

Proposition 2. *If G_m (resp. G_e) contains any over s-rigid sub-graph after updating, then it already contained an over s-rigid sub-graph before updating.*

Proposition 3. *If G_m (or G_e) contains any over s-rigid sub-graph before updating, then it will still contain an over s-rigid sub-graph after updating.*

Proposition 4. *The geometric constraints added to preserve s-rigidity are redundant to the initial constraint system.*

Propositions 1,2 and 3 establish that the s-rigidity of the initial system is preserved after updating. Proposition 4 establishes that the set of solutions of the initial system is preserved after updating.

Notations

K denotes the latest detected cluster, G_e^0 stands for the graph G_e *before* the updating due to detection of K, G_e^1 is the graph G_e *after* updating. G_m^0 denotes the graph G_m *before* updating and G_m^1 is the graph G_m *after* updating. N_K is the vertex replacing the cluster K in G_m^1. K^1 is the updated cluster K in G_e^1, made of the interface objects of K along with interface constraints that make them s-rigid relatively to each other.
R represents the weight of a rigid assembly ($R = (d(d+1)/2$ in dimension d). W_0 denotes the weight of G_m^0 (resp G_e^0) while W_1 represents the weight of G_m^1 (resp G_e^1). Note that $W_0 = W_1 = R$ if these graphs are s-rigid.
w_r represents the sum of the weights of the internal objects in K, w_i is the sum of the weights of the interface objects in K, w_c is the sum of the weights of the constraints in K and w_p is the the sum of the weights of the objects in K that are already part of previous clusters, i.e. shared objects. Note that w_p is part of $w_i + w_r$, since the share objects can be as well internal objects as interface objects.
The fact that K is s-rigid holds the following property: $w_i + w_r - w_c = R$.

Proof of Proposition 1

Updating of G_m:
Proposition 1 holds for G_m. Intuitively, replacing K by N_K duplicates the shared objects of K, which introduces an excess weight in G_m. The addition of coincidence constraints will compensates this excess weight.

More formally, the updating step of G_m is composed of 3 operations:

1. Removal of objects and constraints of K. After this operation, the weight of G_m^1 becomes $W_0 - w_i - w_r + w_p + w_c = W_0 - R + w_p$.
2. Addition of the vertex N_K representing K. this adds a weight R to the weight of G_m^1, which then becomes $W_0 + w_p$.

3. Addition of coincidence constraints. Coincidence constraints are added to G_m between the new vertex representing K and vertices to which K's shared objects already belong. The weight of this constraints is exactly the weight of the shared objects, i.e. w_p. Thus, G_m^1 has weight $W_1 = W_0$. □

Updating of G_e:
Proposition 1 holds for G_e. Intuitively, the removal of the internal objects of K and all the constraints of K from G_e is compensated by the addition of interface constraints. Indeed, the weight of the interface constraints that are added is exactly the weight of the interface objects of K minus R, so that the interface objects along with the interface constraints represent an s-rigid sub-graph.

More formally, the updating step of G_e is divided into 3 operations:

1. Removal of constraints of K. After this operation, the weight of G_e^1 becomes $W_0 + w_c$.
2. Removal of internal objects of K. Then the new weight of G_e^1 is $W_0 + w_c - w_r$.
3. Addition of interface constraints. This reduces the weight of G_e^1 to $W_1 = W_0 + w_c - w_r - w_i + R = W_0$. □

Proof of Proposition 2

Updating of G_m:

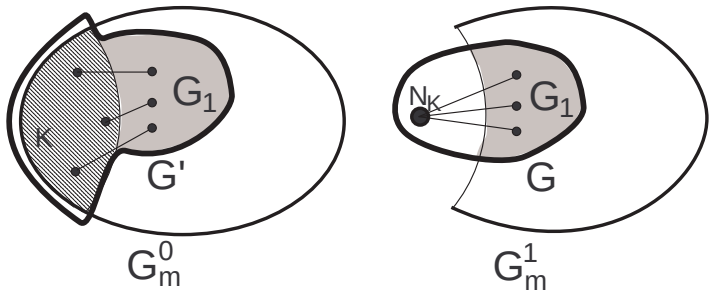

Fig. 4. Proof of Proposition 2 for G_m - G contains N_K

Informally, it is always possible to find one sub-graph G' of G_m^0 which has a weight equal to the weight of a sub-graph G of G_m^1.

Indeed, there are two cases for G:

- G does not contain the vertex N_K. Then $G' = G$ is also a sub-graph of G_m^0.
- G contains the vertex N_K. Let G_1 be the sub-graph of G without this vertex N_K. G_1 is also a sub-graph of G_m^0. Then, G' is the sub-graph of G_m^0 made of G_1 and the whole cluster K. K and N_K have the same weight R. Moreover,

For each object o in K that is shared with a previous cluster C represented by a vertex N_C in G_1, there is a coincidence constraint between N_K and N_C.
It must be understand that a shared object can only be included in a new cluster by a single extension. Indeed, shared objects are interface objects of previous clusters. Interface objects remains only in G_e [4]. Only the extension step is performed in G_e, hence only a single extension can include an interface object of a previous cluster in the current cluster.
To be included by a single extension into the current cluster K, an object o of weight w must share a set of constraints with other objects already in K such that the total weight of these constraints is w.
Hence, for one coincidence constraint of weight w in G_m^1 between N_K and N_C, there was a set of constraints in G_m^0 between objects of K and the vertex N_C which total weight of w also.
This proves that G and G' have the same weight.

There exists a sub-graph G' of G_m^0 which has the same weight as G, a sub-graph of G_m^1. Hence, if G is over s-rigid, G' is also over s-rigid. □

Updating of G_e:

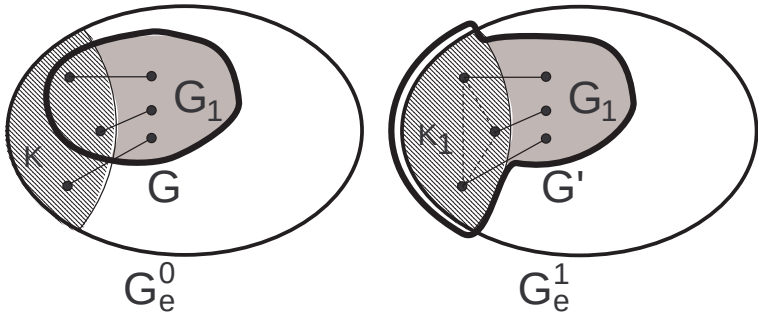

Fig. 5. Proof of Proposition 2 for G_e - G shares a common part with K^1

Informally, it is always possible to find one sub-graph G' of G_e^0 which has a weight lower than or equal to the weight of a sub-graph G of G_e^1.

Indeed there are two cases:

- G contains no interface constraint. Then $G' = G$ is also a sub-graph of G_e^0.
- G contains some of the interface constraints added during the updating step for the cluster K. Hence G contains a subset of the interface objects of K. G_1 is the sub-part of G which does not contain these interface objects of K. Then G_1 is also a sub-graph of G_e^0. Thus, G' is the sub-graph of G_e^0 made of

[4] until they become internal object of a future cluster

G_1 and the cluster K in a whole.
Due to the s-rigidity of K (and K_1), a sub-part of K (or K^1) has a weight greater than or equal to R while K in a whole has a weight exactly equal to R. Hence G' has a weight lower than or equal to G.

There exists a sub-graph G' of G_e^0 which has a weight lower than or equal to the weight of G, a sub-graph of G_e^1. Hence, if G is over s-rigid, G' is also over s-rigid. □

Proof of Proposition 3

Updating of G_m:

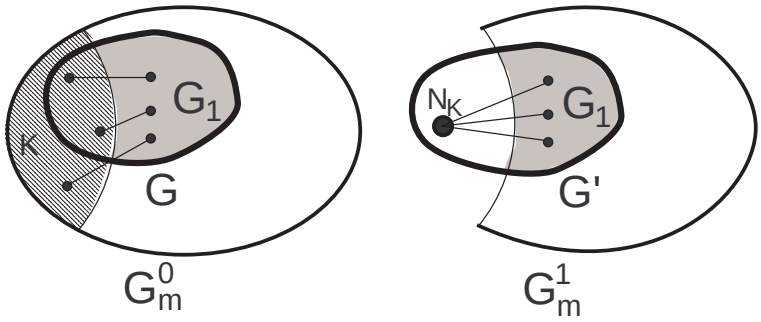

Fig. 6. Proof of Proposition 3 for G_m - G shares a common part with K

Informally, it is always possible to find one sub-graph G' of G_m^1 which has a weight lower than or equal to the weight of a sub-graph G of G_m^0.

Indeed there are two cases:

- G does not contain a sub-part of the cluster K. Then $G' = G$ is also a sub-graph of G_m^1.
- G contains a sub-part of the cluster K. Let G_1 be the sub-graph of G without this sub-part of K, then G_1 is also a sub-graph of G_m^1. G' is the sub-graph of G_m^1 made of G_1 and of the vertex N_K.
  K and N_K have the same weight. Due to the s-rigidity of K, a sub-part of K has a weight greater than or equal to R while K in a whole has a weight exactly equal to R. Hence G' has a weight lower than or equal to G.

There exists a sub-graph G' of G_m^1 which has a weight lower than or equal to the weight of G, a subgraph of G_m^0. Hence if G is over s-rigid, G' is also over s-rigid. □

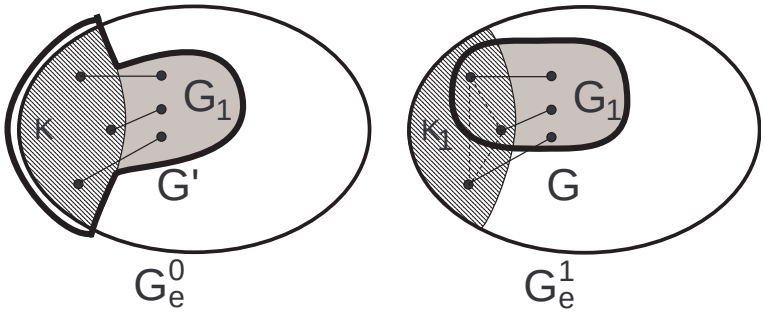

Fig. 7. Proof of Proposition 3 for G_e - G shares a common part with K

Updating of G_e:

Informally, it is always possible to find one sub-graph G' of G_e^1 which has a weight lower than or equal to the weight of a sub-graph G of G_e^0.

Indeed there are two cases:

- G does not contain a sub-part of the cluster K. Then $G' = G$ is also a sub-graph of G_e^1.
- G contains a sub-part of the cluster K. Let G_1 be the sub-part of G which does not contain the sub-part of K. Then G_1 is also a sub-graph of G_e^1. Then, G' is the sub-graph of G_e^1 made of G_1 and the cluster K^1 in a whole (all the interface objects of K and all the interface constraints added to rigidify these interface objects).
  Due to the s-rigidity of K, a sub-part of K has a weight greater than or equal to R while K and K_1 in a whole have a weight exactly equal to R. Hence G' has a weight lower than or equal to G.

There exists a sub-graph G' of G_e^1 which has a weight lower than or equal to the weight of G, a sub-graph of G_e^0. Hence if G is over s-rigid, G' is also over s-rigid. □

Proof of Proposition 4
As described in section 4, a sub-system to solve always corresponds to a sub-graph of G_e. So we will only have to prove that interface constraints are added redundantly to the initial system. This is done by construction: the interface constraints that replaces a cluster K are designed to provide the same set of configurations as the initial constraints of K (see section 4 - Interface Constraints). □

6 Conclusion

The new heuristic introduced in this paper for the planning phase produces in general smaller blocks for the solving phase, which is interesting despite the addi-

tion of constraints to the system and the duplication of some objects. Moreover, interval propagation techniques, which are general and complete solving techniques, can efficiently make use of the produced construction plan to compute all possible positions and orientations of the geometric objects under constraints. The reader will refer to [JTNR00] for more details about this solving phase.

References

[BFH+95] William Bouma, Ioannis Fudos, Christoph Hoffmann, Jiazhen Cai, and Robert Paige. Geometric constraint solver. *Computer Aided Design*, 27(6):487–501, 1995.

[DMS98] Jean-François Dufourd, Pascal Mathis, and Pascal Schreck. Geometric construction by assembling subfigures. *Artificial Intelligence*, 99:73–119, 1998.

[FH93] Iaonnis Fudos and Christoph Hoffmann. Correctness proof of a geometric constraint solver. Technical Report TR-CSD-93-076, Purdue University, West Lafayette, Indiana, 1993.

[FH97] Iaonnis Fudos and Christoph Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, 16(2):179–216, 1997.

[Hen92] Bruce Hendrickson. Conditions for unique realizations. *SIAM J Computing*, 21(1):65–84, 1992.

[HLS97] Christoph Hoffmann, Andrew Lomonosov, and Meera Sitharam. Finding solvable subsets of constraint graphs. In *Proc. Constraint Programming CP'97*, pages 463–477, 1997.

[HLS98] Christoph Hoffmann, Andrew Lomonosov, and Meera Sitharam. Geometric constraint decomposition. In B. Brüderlin and D. Roller, editors, *Geometric Constraint Solving and Applications*, pages 170–195. Springer, 1998.

[JTNR00] Christophe Jermann, Gilles Trombettoni, Bertrand Neveu, and Michel Rueher. A constraint programming approach for solving rigid geometric systems. Technical Report 00-43, University of Nice, France, 2000.

[Lam70] G. Laman. On graphs and rigidity of plane skeletal structures. *J. Eng. Math.*, 4:331–340, 1970.

[Lee98] Jae Yeol Lee. A 2d geometric constraint solver for parametric design using graph analysis and reduction. In *Automated Deduction in Geometry, ADG'98*, volume 1669 of *LNAI*, pages 258–274. Springer, 1998.

[LM98] Hervé Lamure and Dominique Michelucci. Qualitative study of geometric constraints. In Beat Bruderlin and Dieter Roller, editors, *Geometric Constraint Solving and Applications*, pages 234–258. Springer, 1998.

[VSR92] A. Verroust, F. Schonek, and D. Roller. Rule oriented method for parametrized computer aided design. *Computer Aided Design*, 24(6):531–540, 1992.