Algorithmes pour la détection de rigidités dans les CSP géométriques

Christophe Jermann
LINA, BP 92208, 44322 Nantes Cedex 3, France
e-mail : Christophe.Jermann@irin.univ-nantes.fr

Bertrand Neveu **Gilles Trombettoni**
Équipe COPRIN, CERTIS-I3S-INRIA
INRIA, BP 93, 06902 Sophia Antipolis Cedex, France
e-mail : Bertrand.Neveu@sophia.inria.fr
Gilles.Trombettoni@sophia.inria.fr

Résumé

Le théorème de Laman permet de caractériser la rigidité des systèmes à barres en 2D. La rigidité structurelle est basée sur une généralisation de ce théorème. Elle est généralement considérée comme une bonne heuristique pour identifier des sous-parties rigides dans les CSP géométriques (GCSP), mais peut en réalité se tromper sur des sous-systèmes très simples car elle ne tient pas compte des propriétés géométriques vérifiées par les objets. Hoffmann *et al.* ont proposé en 1997 des algorithmes à base de flots s'appuyant sur la caractérisation par rigidité structurelle pour répondre aux principales questions liées au concept de rigidité : déterminer si un GCSP est rigide, identifier ses composantes rigide, sur-rigide et sous-rigide, en minimiser la taille, *etc*.

La rigidité structurelle étendue, une nouvelle caractérisation de la rigidité, a été proposée par Jermann *et al.* en 2002. Elle permet de prendre en compte les propriétés géométriques du GCSP étudié et s'avère ainsi plus fiable. Dans le présent article, nous présentons des algorithmes qui répondent aux principales questions liées à la notion de rigidité en utilisant cette nouvelle caractérisation. Plus précisément, nous montrons que deux modifications de la fonction de distribution de flot utilisée dans les algorithmes de Hoffmann *et al.* permettent l'obtention d'une famille d'algorithmes basés sur la rigidité structurelle étendue. Nous démontrons la correction et la complétude des nouveaux algorithmes et étudions leur complexité en pire cas.

1 Introduction

Les problèmes géométriques apparaissent dans de nombreuses applications pratiques : la CAO, la robotique et la biologie moléculaire en sont trois exemples. Le paradigme de la programmation par contraintes (PPC) consiste à modéliser les problèmes

de façon déclarative sous forme de problèmes de satisfaction de contraintes (CSP) et à utiliser des outils génériques pour résoudre ces CSP. L'application de ce paradigme aux problèmes géométriques permet de les considérer comme des problèmes de satisfaction de contraintes géométriques (GCSP).

Dans un GCSP, les contraintes (distances, angles, incidences, ...) visent à restreindre les positions, orientations et dimensions que peuvent adopter des objets géométriques (points, droites, plans, ...). La résolution d'un GCSP permet donc de déterminer des positions, orientations et dimensions de tous les objets qui satisfont les contraintes. Cependant, des questions d'ordre qualitatif peuvent se poser avant de résoudre un GCSP: le système modélisé est-il déformable ? Si tel est le cas, quelles sont les déformations qu'il admet ? Admet-il des solutions, et sinon pourquoi ? Ces questions apparaissent souvent dans les domaines que nous avons cités, où un concepteur agissant plus au niveau géométrique qu'au niveau CSP, souhaite connaître a priori les propriétés du système géométrique qu'il a modélisé.

On a alors recours au concept géométrique de rigidité et à différentes caractérisations de ce concept pour essayer de répondre à ces questions. Intuitivement, les sous-parties rigides d'un GCSP sont indéformables, alors que ses sous-parties sous-rigides (sous-déterminées) présentent des déformations et ses sous-parties sur-rigides (sur-déterminées) fournissent des explications à l'absence de solution globale.

Ces informations qualitatives sont également souvent utilisées de façon implicite ou explicite dans les résolveurs dédiés aux GCSP [Kra92; BFH+95; DMS98; LM98; HLS01; JTNR00]. En particulier, les méthodes de décomposition géométriques ont pour but de produire des séquences de sous-GCSP rigides pouvant être résolus séparément puis assemblés. Elles nécessitent des algorithmes efficaces permettant d'identifier de petits sous-GCSP rigides dont les solutions partielles peuvent être assemblées pour reconstituer une solution globale du GCSP initial.

Le concept de rigidité dispose donc d'un statut central dans l'analyse et la résolution efficace de GCSP. La principale problématique est alors de concevoir des caractérisations de ce concept qui soient suffisamment fiables et puissent donner naissance à des algorithmes efficaces répondant aux questions : un GCSP est-il rigide ? Quelles sont ses sous-parties rigide, sur-rigide et sous-rigide ? Peut-on identifier de telles sous-parties de taille minimale ?

Les méthodes de caractérisation de rigidité peuvent être classées en deux catégories : les *approches à base de règles* [BFH+95; Kra92] utilisent un répertoire de formes rigides connues qui ne peut couvrir tous les cas de figure, alors que les *approches structurelles* [HLS97; LM98] utilisent des algorithmes de flots (ou de couplage maximum) pour vérifier une propriété appelée *rigidité structurelle*, basée sur un compte des degrés de liberté dans le GCSP.

Les approches structurelles sont plus générales puisqu'elles peuvent être appliquées à toute classe de GCSP (tous types d'objets, tous types de contraintes). Cependant, la rigidité structurelle n'est qu'une approximation de la rigidité dans le cas général, c-à-d. qu'il existe des GCSP mal caractérisés par cette propriété. Plusieurs heuristiques sont alors employées pour assister cette propriété, mais aucune ne permet d'éliminer tous les cas d'erreur.

Dans [JNT02], nous avons proposé une nouvelle caractérisation de la rigidité, appelée *rigidité structurelle étendue*, qui subsume la rigidité structurelle, et correspond

même exactement à la rigidité pour des GCSP exempts de contraintes redondantes.

Dans le présent article, nous proposons de nouveaux algorithmes basés sur notre nouvelle caractérisation de la rigidité pour répondre aux principales questions liées au concept de rigidité. Ces nouveaux algorithmes découlent de deux modifications principales dans la fonction de distribution de flots utilisée par les algorithmes de Hoffmann *et al.* [HLS97].

Après les définitions données en section 2, nous introduisons le principe de la caractérisation de la rigidité à base de flots et présentons brièvement les algorithmes proposés par Hoffmann *et al.* (section 3). Finalement, nous présentons les deux modifications que nous proposons dans la fonction `Distribute` et la famille d'algorithmes qui en découle (section 4). Nous analysons les propriétés de chacun des nouveaux algorithmes proposés: correction, complétude et complexité.

2 Définitions

Dans cette section sont énoncées et expliquées les définitions nécessaires à la compréhension de cet article.

2.1 Problème de satisfaction de contraintes géométriques

Définition 1 GCSP
Un **problème de satisfaction de contraintes géométriques** *(GCSP)* $S = (O,C)$ *est composé d'un ensemble* O *d'objets géométriques et d'un ensemble* C *de contraintes géométriques.*
$S' = (O',C')$ *est un* **sous-GCSP** *de* $S = (O,C)$*, noté* $S' \subset S$*, ssi* $O' \subset O$ *et* $C' = \{c \in C | c$ *ne porte que sur des objets dans* $O'\}$*, c-à-d. que* S' *est induit par* O'*.*

Les objets géométriques usuels sont les points, les droites et, en 3D, les plans. On peut aussi considérer des objets géométriques plus complexes, tels que des cercles, des coniques, des parallélépipèdes, *etc*. Les paramètres d'un objet géométrique définissent sa position, son orientation et ses dimensions.

Les contraintes géométriques sont, par exemple, des distances, des angles, des incidences, des parallélismes, des symétries, des alignements, *etc*. Les contraintes ont pour effet de restreindre l'ensemble des positions, orientations et dimensions que peuvent prendre les objets géométriques du GCSP.

Une solution d'un GCSP est la donnée d'une position, d'une orientation et d'un jeu de dimensions pour chaque objet géométrique de telle sorte que toutes les contraintes géométriques soient satisfaites.

La figure 1-a présente un GCSP en 2D constitué de 3 droites liées par 2 parallélismes et deux distances droite-droite. La figure 1-b présente un GCSP en 3D constitué d'une droite et de 5 points, liés par 4 incidences point-droite et 5 distances point-point.

Pour être résolu, un GCSP est généralement transformé en un système d'équations, chaque objet étant représenté par des variables définissant sa position et son orientation, et chaque contrainte devenant un sous-système d'équations sur les variables des objets qu'elle contraint.

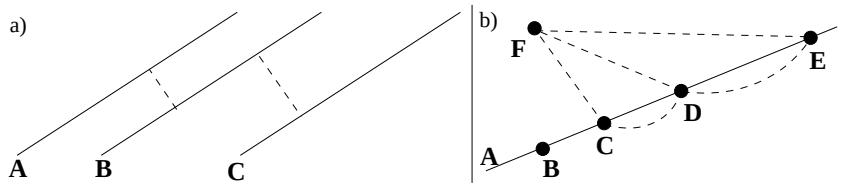

FIG. 1 – *Deux exemples de GCSP*

Hypothèses : Nous supposerons que les objets géométriques sont indéformables (pas de cercle à rayon variable par exemple) et que les contraintes ne peuvent porter que sur les positions et orientations relatives des objets (pas de fixation par rapport au repère global par exemple). Ces limitations simplifient la définition des caractérisations structurelles de la rigidité et sont nécessaires aux méthodes de résolution de GCSP basées sur la rigidité.

Sous ces hypothèses, une solution d'un GCSP est composée d'une position et d'une orientation pour chacun de ses objets satisfaisant toutes ses contraintes.

2.2 Rigidité

La rigidité d'un GCSP se définit à partir des mouvements que celui-ci admet[1]. On distingue deux types de mouvements : les déformations, qui ne préservent pas les positions et orientations relatives des objets, et les déplacements (rotations et translations) qui les préservent. Intuitivement, un GCSP est **rigide** s'il n'admet aucune déformation et admet tous les déplacements de l'espace géométrique considéré[2]. Un GCSP qui admet des déformations est dit **sous-rigide**, alors qu'un GCSP n'admettant pas certains déplacements, ou aucune solution, est dit **sur-rigide**. Des définitions plus formelles peuvent être trouvées dans [Jer02].

Dans l'exemple présenté en figure 1-b, le sous-GCSP CDF est rigide : un triangle est indéformable et admet tous les déplacements de l'espace ; AF est sous-rigide puisque la droite A et le point F ne sont liés par aucune contrainte et peuvent donc être déplacés indépendamment l'un de l'autre ; le sous-GCSP $ACDEF$ est quant à lui sur-rigide : il est génériquement impossible de placer le point F à l'intersection des 3 sphères (contraintes de distance) dont les centres C, D et E sont alignés.

2.3 Rigidité structurelle

La **rigidité structurelle** correspond à une analyse des **degrés de liberté** (DDL) dans le GCSP. Intuitivement, un DDL représente un mouvement indépendant dans le

1. En réalité, la rigidité et les mouvements se définissent au niveau de chaque solution d'un GCSP. Cependant, comme c'est généralement le cas en CAO, on considère la rigidité au niveau du GCSP comme représentant celle de toutes ses solutions car on souhaite généralement étudier des systèmes non résolus afin d'en déduire des propriétés générales (c-à-d., valables pour toute solution).

2. Cette seconde condition est toujours vérifiée sous l'hypothèse que nous avons posée sur les contraintes géométriques.

GCSP. Plus formellement :

Définition 2 Degré de liberté (DDL)
- Objet o : DDL(o) est le nombre de variables indépendantes définissant la position et l'orientation de o.
- Contrainte c : DDL(c) est le nombre d'équations indépendantes représentant la contrainte c.
- GCSP $S = (O,C)$: DDL(S)=$\sum_O$DDL(o) - $\sum_C$DDL(c).

Le tableau ci dessous présente le nombre de degrés de liberté de quelques objets et contraintes géométriques usuels en 2D et 3D :

Dim.	Objets	DDL	Contraintes	DDL
2D	Point	2	Distance Point-Point	1
	Droite	2	Angle Droite-Droite	1
	Cercle	2	Parallélisme Droite-Droite	1
			Incidence Point-Droite	1
3D	Point	3	Distance Point-Point	1
	Droite	4	Angle Droite-Droite	1
	Plan	3	Parallélisme Plan-Plan	1
	Cercle	5	Parallélisme Droite-Droite	2
	Sphère	3	Incidence Point-Droite	2
			Incidence Point-Plan	1

A partir de ces tableaux, on peut calculer le nombre de DDL de quelques sous-GCSP de la figure 1-b. Par exemple, le sous-GCSP ACD est constitué d'une droite et de 2 points, liés par 2 incidences point-droite et 1 distance point-point; il totalise donc $4 + 3 + 3 - 2 - 2 - 1 = 5$ DDL. Egalement, les sous-GCSP CDF et AF issus de la même figure totalisent respectivement 6 et 7 DDL.

La rigidité structurelle est une généralisation du théorème de Laman [Lam70] qui caractérise la rigidité générique des systèmes à barres en 2D. Elle est basée sur l'intuition suivante : si un GCSP admet moins (resp. plus) de mouvements que le nombre de déplacements (d translations et $\frac{d(d-1)}{2}$ rotations en dimension d, soit $\frac{d(d+1)}{2}$ déplacements indépendants en tout) admis par l'espace géométrique qui le contient, alors il est sur- (resp. sous-)rigide. Plus formellement :

Définition 3 Rigidité structurelle (s_rigidité)
Un GCSP $S = (O,C)$ en dimension d est s_rigide *ssi DDL(S) = $\frac{d(d+1)}{2}$ et S ne contient pas de sous-GCSP sur-s_rigide.*
S est sous-s_rigide *ssi DDL(S) > $\frac{d(d+1)}{2}$ et S ne contient pas de sous-GCSP sur-s_rigide.*
S est sur-s_rigide *ssi $\exists S' \subseteq S$ tel que DDL(S') < $\frac{d(d+1)}{2}$.*

En pratique, la rigidité structurelle est considérée comme une bonne approximation de la rigidité [LM98; HLS97]. Cependant, l'écart entre s_rigidité et rigidité est important (cf. [JNT02; Jer02]). Nous illustrons ici cet écart sur 2 sous-GCSP issus de la figure 1-b : $ABCD$ est s_rigide en 3D puisque DDL($ABCD$)=6 ($\frac{d(d+1)}{2} = 6$ en

dimension $d = 3$); cependant, ce sous-GCSP est en réalité sous-rigide : le point B peut bouger indépendamment des points C et D sur la droite A. $ACDE$ est quant à lui sur-s_rigide car DDL($ACDE$)=5, mais il s'agit en réalité d'un sous-GCSP rigide.

2.4 Rigidité structurelle étendue

Dim.	GCSP	DDR
2D	Deux points incidents (c-à-d. confondus)	2
	Deux points non-incidents	3
	Deux droites sécantes (c-à-d. non-parallèles)	3
	Deux droites parallèles	2
	Un point et une droite non-incidents	3
	Un point et une droite incidents	3
3D	Deux points incidents (c-à-d. confondus)	3
	Deux points non-incidents	5
	Deux droites sécantes (c-à-d. coplanaires)	6
	Deux droites non-sécantes et non-parallèles	6
	Deux droites parallèles	5
	Un point et une droite non-incidents	6
	Un point et une droite incidents	5
	Un point et un plan non-incidents	5
	Un point et un plan incidents	5
	Une droite et un plan non incidents	6
	Une droite et un plan incidents	5
	Deux plans parallèles	3
	Deux plans sécants	5

La rigidité structurelle étendue (notée es_rigidité) est basée sur le concept de *degré de rigidité* (DDR). Le degré de rigidité d'un GCSP représente le nombre de déplacements admis par celui-ci. Ce nombre dépend des *relations géométriques* vérifiées par les objets du GCSP. Les relations géométriques qui ont une influence sur le degré de rigidité correspondent à des positions ou orientations relatives spécifiques des objets géométriques; parallélisme, orthogonalité, incidence, tangence, sont autant de relations qui, si elles sont vérifiées, font varier le degré de rigidité. Par exemple, en 2D, deux droites admettent généralement 3 déplacements (2 translations et 1 rotation) indépendants et disposent donc d'un degré de rigidité égal à 3; cependant, si ces droites sont parallèles, alors la translation selon la direction de ces droites représente l'identité pour ces droite, c-a-d. que cette translation est sans effet. Deux droites parallèles en 2D disposent donc seulement de 2 déplacements indépendants, et ont donc pour degré de rigidité 2. Le tableau ci-dessus décrit le DDR associé à quelques petits GCSP en 2D et 3D.

Notons que les propriétés géométriques qui ont une influence sur le DDR correspondent à des contraintes de positionnement relatifs des objets géométriques: parallélisme, orthogonalité, incidence, ... ces propriétés peuvent donc apparaître explicitement dans le GCSP étudié, sous la forme d'une contrainte. Elle peuvent toutefois également être induites par un sous-ensemble des contraintes du GCSP considéré, et

être ainsi implicites. Par exemple, 2 droites parallèles à une même troisième sont parallèles entre elles en 2D : 2 contraintes explicites de parallélisme peuvent ainsi en impliquer une troisième, implicite. Dans ce dernier cas, inférer ces propriétés (et donc déterminer le DDR) est équivalent à la preuve de théorème géométrique.

Le principe de la rigidité structurelle étendue est de comparer le nombre de DDL d'un GCSP au DDR de ce même GCSP, c-à-d. le nombre de mouvements au nombre de déplacements. On peut ainsi déterminer si un GCSP admet ou non des déformations.

Définition 4 Rigidité structurelle étendue (es_rigidité)
Un GCSP $S = (O,C)$ *est* es_rigide *ssi DDL(S)=DDR(S) et* S *n'est pas sur-es_rigide.*
S *est* sous-es_rigide *ssi DDL(S)* $>$ *DDR(S) et* S *n'est pas sur-es_rigide.*
S *est* sur-es_rigide *ssi* $\exists S' \subseteq S$, *DDL(*S'*)*$<$ *DDR(*S'*).*

Le tableau ci-dessous présente une comparaison de la rigidité, la s_rigidité et la es_rigidité sur quelques sous-GCSP issus de la figure 1-b. Dans ce tableau, les cas de divergence entre caractérisations structurelles (s_rigidité et es_rigidité) et véritable rigidité sont signalés en gras. Notons que la s_rigidité utilisée pour cette comparaison est assistée d'une heuristique courante qui consiste à ne considérer que les sous-GCSP contenant au moins $d+1$ objets géométriques en dimension d; sans l'usage de cette heuristique, la caractérisation serait entièrement fausse. Les cases contenant *bien* signifient rigide, s_rigide ou es_rigide selon la colonne où elles apparaissent.

Sous-GCSP	Rigidité	DDL	s_rigidité	DDR	es_rigidité
ABCD	sous	6	**bien**	5	sous
ACDE	bien	5	**sur**	5	bien
ABCDF	sous	7	sous	6	sous
ACDEF	sur	5	sur	6	sur
ABCDEF	sur	6	sur	6	sur

On peut constater sur ce tableau que la s_rigidité ne correspond pas toujours à la rigidité, alors que la es_rigidité lui correspond pour chacun de ces sous-GCSP. En fait, la es_rigidité correspond à la rigidité pour tout sous-GCSP issu des figures 1-a ou 1-b.

De façon générale, la es_rigidité subsume la s_rigidité. Intuitivement, ceci s'explique par le fait que lorsque le DDR d'un sous-GCSP en dimension d vaut $\frac{d(d+1)}{2}$, les définitions de ces deux caractérisations sont équivalentes; alors que lorsque le DDR est différent de $\frac{d(d+1)}{2}$, la s_rigidité utilise une mauvaise estimation du nombre de déplacements admis par le sous-GCSP est parvient donc à une mauvaise conclusion.

La rigidité structurelle étendue demeure toutefois une approximation de la rigidité dans le cas général: elle peut être trompée par la présence de contraintes redondantes dans un GCSP. Elle pose également le problème du calcul du DDR, qui requiert la détermination des propriétés géométriques (incidences, parallélismes, ...) induites par les contraintes du GCSP et équivaut généralement à la preuve de théorème en géométrie. Nous renvoyons le lecteur à [JNT02] pour plus de détails sur cette nouvelle caractérisation et une comparaison plus complète avec la rigidité structurelle.

3 Caractérisation par rigidité structurelle

Dans cette section, nous allons présenter les travaux de Hoffmann *et al.* sur la caractérisation algorithmique de la rigidité structurelle par des mécanismes de flots. Cette caractérisation s'appuie sur une représentation des GCSP sous forme de réseaux dans lesquels une distribution de flots correspond à une distribution des degrés de liberté des contraintes sur les degrés de liberté des objets. Ce réseau fut introduit dans [HLS97], ainsi que la fonction `Distribute` qui est à la base des algorithmes `Dense` et `Minimal_Dense` utilisés respectivement pour identifier des sous-GCSP s_rigide ou sur-s_rigide et en minimiser le nombre d'objets.

3.1 Réseau Objet-Contrainte

Un GCSP $S = (O,C)$ peut être représenté par un réseau $G = (s,V,t,E,w)$ appelé *réseau objets-contraintes*. La figure 2-a représente le réseau objets-contraintes correspondant au GCSP de la figure 1-a.

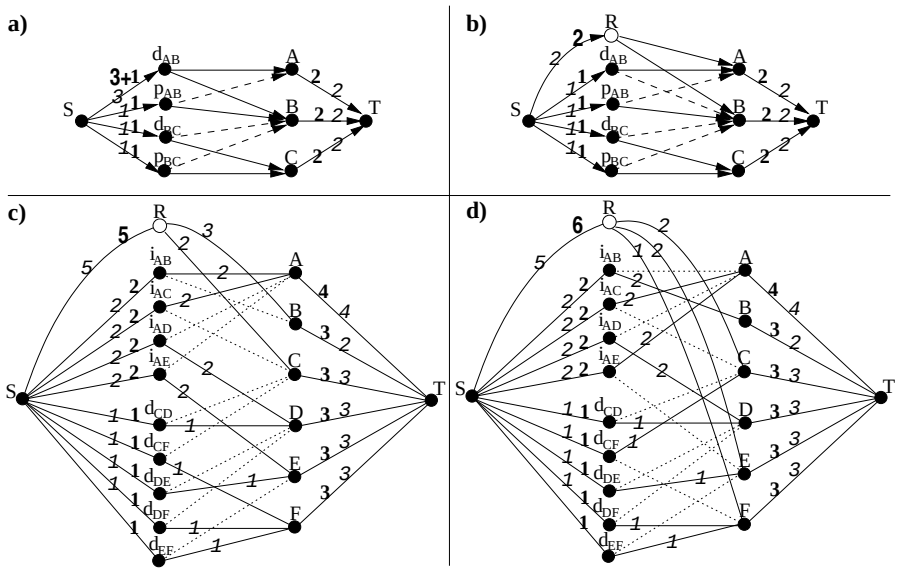

FIG. 2 – *Réseaux objets-contraintes et distributions de flots par les algorithmes* `Dense` *et* `Over-Rigid`

Définition 5 Réseau objets-contraintes (s,V,t,E,w)
- s est la source et t est le puits.
- Chaque objet $o \in O$ est représenté par un nœud-objet $v_o \in V$.
- Chaque contrainte $c \in C$ est représentée par un nœud-contrainte $v_c \in V$.
- Pour chaque objet $o \in O$, on ajoute un arc $(v_o \rightarrow t) \in E$ de capacité $w(v_o \rightarrow t)$ =DDL(o).

- Pour chaque contrainte $c \in C$, on ajoute un arc $(s \rightarrow v_c) \in E$ de capacité $w(s \rightarrow v_c)$ =DDL(c).
- Pour chaque objet $o \in O$ sur lequel porte une contrainte $c \in C$, il existe un arc $v_c \rightarrow v_o$ de capacité $w(v_c \rightarrow v_o) = \infty$ dans E.

Par définition, un flot dans ce réseau représente une distribution des DDL des contraintes sur les DDL des objets ; c'est ce principe qui est utilisé pour la caractérisation structurelle de la rigidité.

3.2 Principe de caractérisation par un flot

Du point de vue géométrique, la caractérisation de la rigidité s'effectue en vérifiant qu'un GCSP n'admet que des déplacements. Sous cet angle, la détection de rigidité à base de flots peut être expliquée comme suit :

1. retirer K déplacements du GCSP en introduisant K DDL supplémentaires du côté des contraintes ;
2. vérifier si un sous-GCSP sur-contraint S' existe en calculant un flot maximum dans le réseau objets-contraintes surchargé ;
3. si tel est le cas, alors S' vérifie DDL(S')$< K$.

En effet, nous avons déjà expliqué qu'un flot dans le réseau objets-contraintes représente une distribution des DDL des contraintes sur les DDL des objets, ce qui revient intuitivement à choisir quelles contraintes résolvent quels objets. Un flot maximum représente donc une distribution *optimale* des DDL dans le GCSP. Si un flot maximum ne peut saturer tous les arcs issus de la source du réseau, cela signifie que certains DDL des contraintes ne peuvent être distribués sur les objets (cf. définition 5). Il existe alors obligatoirement un sous-GCSP S' sur-contraint (c-à-d. DDL(S')<0) dans le GCSP.

La détection structurelle de rigidité exploite cette propriété pour identifier un sous-GCSP S' qui vérifie DDL(S')$<$ K pour une valeur K donnée. Pour ce faire, K DDL additionnels sont artificiellement introduits dans le réseau en provenance de la source. Selon le principe expliqué ci-dessus, si le calcul d'un flot maximum ne permet pas de saturer tous les arcs issus de la source cela signifie que les objets ne peuvent pas *absorber* les DDL des contraintes *plus ces K DDL additionnels*. S' vérifie alors nécessairement DDL(S')$<$ K. [HLS97] ont montré que S' est alors induit par l'ensemble des nœuds-objets traversés durant la dernière recherche d'un chemin augmentant dans le réseau : S' est induit par l'ensemble de tous les nœuds-objets atteignables par un parcours du graphe résiduel partant de la source.

En choisissant la valeur adéquate pour K, [HLS97] ont dérivé des algorithmes pour identifier des sous-GCSP s_rigides ($K = \frac{d(d+1)}{2} + 1$) ou sur-s_rigides ($K = \frac{d(d+1)}{2}$).

3.3 Fonction `Distribute`

La fonction `Distribute` est la mise en application algorithmique du principe de caractérisation structurelle de la rigidité à base de flots. Cette fonction, proposée

dans [HLS97], applique la surcharge K en augmentant simplement la capacité de l'arc liant la source à une contrainte c dans le réseau, c-à-d. en ajoutant K au nombre de DDL de la contrainte c. Ceci a pour but de retirer K déplacements au sous-GCSP induit par les objets liés à c.

Algorithme 1 Distribute (S : un GCSP ; K : un entier ; c : une contrainte) **retourne** S' : un GCSP

Requiert: $K > 0$, $S = (O,C)$, $c \in C$

Assure: $S' \subset S$ vérifie DDL(S')$< K$, ou S' est vide

$G \leftarrow$ `Réseau-Surchargé`(S,K,c)

$V \leftarrow$ `Ford-Fulkerson`(G)

$S' \leftarrow$ `Sous-GCSP-Induit`(V,S)

Retourne S'

La fonction `Réseau-Surchargé` retourne le réseau objets-contraintes correspondant au GCSP S dans lequel la contrainte c a été surchargée, c-à-d. que l'arc $s \rightarrow c$ liant la source s à la contrainte c est de capacité DDL(c)+K. Le calcul du flot maximum dans ce réseau est effectué par la fonction `Ford-Fulkerson` qui applique l'algorithme classique de [FF62]. Cette fonction retourne l'ensemble V des nœuds-objets traversés lors de la dernière recherche d'un chemin augmentant, cet ensemble étant vide si le flot maximum sature tous les arcs issus de la source s. Enfin, la fonction `Sous-GCSP-Induit` retourne le sous-GCSP S' induit par V. Selon la preuve de Hoffmann *et al.*, S' vérifie alors DDL(S')$< K$ ou S' est vide.

Exemple d'application de `Distribute`: La figure 2-a illustre l'appel `Distribute(`S,3,dAB`)` sur le GCSP S de la figure 1-a. Un flot maximum est représenté sur le réseau par les chiffres en gras. Comme l'arc $s \rightarrow dAB$ n'est pas saturé dans ce flot maximum, la fonction retourne le sous-GCSP induit par le sous-ensemble d'objets atteignables dans le graphe résiduel en partant de dAB; il s'agit du sous-GCSP AB, qui dispose effectivement de moins de $K = 3$ DDL.

Complexité de `Distribute`: La complexité de la fonction `Distribute` est dominée par celle de la fonction `FordFulkerson` qui effectue le calcul du flot maximum. En effet, la production du réseau objets-contraintes surchargé et l'extraction du sous-GCSP induit par V peuvent être effectués en temps linéaire, alors que le calcul d'un flot maximum est en $O(N^2 * (N + M))$, N étant le nombre de nœuds et M le nombre d'arcs dans le réseau. Notons n le nombre d'objets géométriques et m le nombre de contraintes géométriques; en supposant que les contraintes géométriques considérées sont d'arité bornée[3], on obtient $N = n + m$ et $M \sim n$. Il en résulte que la complexité de la fonction `Distribute` exprimée en nombre d'objets et de contraintes est $O(n * (n + m)^2)$.

3. En pratique, les contraintes géométriques sont toujours d'arité bornée; même une contrainte d'égalité de distance sera d'arité au plus 4.

3.4 Algorithmes de Hoffmann *et al.*

`Dense` et `Minimal-Dense` sont deux algorithmes proposés par [HLS97] qui reposent entièrement sur la fonction `Distribute` et permettent respectivement de caractériser des sous-GCSP s_rigide ou sur-s_rigide et d'en minimiser la taille pour l'opération d'inclusion (cf. définition 1).

3.4.1 Algorithme `Dense`

L'algorithme `Dense` a pour but d'identifier un sous-GCSP sur-s_rigide s'il en existe un dans le GCSP passé en paramètre. Pour ce faire, il consiste simplement à appliquer séquentiellement la fonction Distribute à chaque contrainte du GCSP considéré, avec une surcharge valant à chaque fois $K = \frac{d(d+1)}{2}$. La séquence se termine lorsque l'appel courant retourne un sous-GCSP S' non vide. Par définition de la fonction `Distribute`, S' vérifie alors DDL(S')$< \frac{d(d+1)}{2}$, c-à-d. que S' est sur-s_rigide (cf. définition 3). Notons que cet algorithme peut être utilisé pour identifier des sous-GCSP s_rigide ou sur-s_rigides en changeant simplement la valeur de la surcharge en $K = \frac{d(d+1)}{2} + 1$.

Algorithme 2 Dense ($S = (O,C)$: un GCSP ; d : un entier) **retourne** S' : un GCSP)

Requiert: $d > 0$ représente la dimension de l'espace géométrique contenant S

Assure: $S' \subset S$ vérifie DDL(S')$< \frac{d(d+1)}{2}$, ou S' est vide

$Contraintes \leftarrow C$

$S' \leftarrow EmptyGCSP$

Tant que $Contraintes \neq \emptyset \wedge S' = EmptyGCSP$ **Faire**

$c \leftarrow$ `Pop(`$Contraintes$`)`

$S' \leftarrow$ `Distribute(`$S, \frac{d(d+1)}{2}, c$`)`

Fin Tant que

Retourne S'

Exemple d'application de `Dense` : La figure 2-a illustre aussi l'application de l'algorithme `Dense` sur le GCSP de la figure 1-a : `Dense(`S,2`)`. En effet, cet algorithme va immédiatement effectuer l'appel `Distribute(`S,4,dAB`)`. Comme nous l'avons vu précédemment, `Distribute(`S,3,dAB`)` retourne le sous-GCSP AB, et il en va de même pour l'algorithme `Dense`: la surcharge $K = 4$ étant supérieure à $K = 3$, le flot maximum ne pourra pas non plus saturer l'arc $s \rightarrow dAB$ liant la source à la contrainte considérée.

Le sous-GCSP AB est donc identifié s_rigide ou sur-s_rigide. Il est en réalité sur-s_rigide, mais il s'agit de l'un des cas où la s_rigidité caractérise mal la rigidité, comme nous l'avons vu à la section 2.3.

Complexité de `Dense` : La complexité de l'algorithme `Dense` dépend directement de celle de la fonction `Distribute` puisque cet algorithme consiste à appeler cette

fonction au plus une fois par contrainte du GCSP. Il est donc en $O(m * n * (n + m)^2)$. Il est à noter que l'algorithme présenté dans [HLS97] s'applique sur un réseau construit incrémentalement, ce qui conduit à une complexité pratique moindre puisque l'algorithme termine toujours au plus tôt. Par ailleurs, d'autres optimisations peuvent encore abaisser le coût de cet algorithme. Par exemple, pré-calculer un flot maximum du réseau non-surchargé une seule fois avant le premier appel à l'algorithme `Dense` permet d'avoir seulement à mettre à jour ce réseau pour chaque surcharge introduite, ce qui abaisse la complexité de la fonction FordFulkerson d'un facteur linéaire.

3.4.2 Algorithme `Minimal-Dense`

L'algorithme `Minimal-Dense` a pour but de retourner un sous-GCSP sur-s_rigide minimal au sens de l'inclusion, c-à-d. ne contenant aucun sous-GCSP sur-s_rigide. Pour ce faire, l'algorithme commence par appliquer l'algorithme `Dense` afin d'identifier un sous-GCSP S' sur-s_rigide. Il applique ensuite une étape de minimisation qui consiste à essayer de retirer les objets de S' un à un. Après le retrait d'un objet, un appel à l'algorithme `Dense` permet de tester s'il existe toujours un sous-GCSP sur-s_rigide, auquel cas le processus se poursuit sur ce nouveau sous-GCSP strictement plus petit; autrement, l'objet retiré était nécessaire à l'existence d'un sous-GCSP sur-s_rigide et doit donc être conservé. Notons là encore que l'emploi de l'algorithme `Dense` avec surcharge $K = \frac{d(d+1)}{2} + 1$ conduit à un algorithme `Minimal-Dense` qui identifie et minimise un sous-GCSP s_rigide ou sur-s_rigide.

Exemple d'application de `Minimal-Dense`: appliqué au GCSP de la figure 1-a, cet algorithme retournera également le sous-GCSP AB. En effet, le premier appel à l'algorithme `Dense` retournera AB. Le retrait de A donne alors un sous-GCSP vide de contraintes, qui n'est donc pas s_rigide ou sur-s_rigide; le retrait de B également. A et B étant nécessaires, le sous-GCSP minimisé est bien AB.

Complexité de `Minimal-Dense`: La complexité de l'algorithme dépend directement de celle de l'algorithme `Dense` puisqu'il consiste à appliquer au plus une fois cet algorithme pour chaque objet du sous-GCSP identifié par le premier appel à `Dense`, qui est au pire le GCSP initial. Elle est en $O(n*m*n*(n+m)^2)$. Là encore, les optimisations de l'algorithme `Dense` permettent d'obtenir une complexité pratique inférieure à cette complexité en pire cas.

4 Nouveaux Algorithmes

Nos nouveaux algorithmes, tout comme les algorithmes proposés par [HLS97], sont basés sur un calcul de flot maximum dans le réseau objets-contraintes. Cependant, ils présentent deux différences principales :

- ils utilisent la es_rigidité au lieu de la s_rigidité ;
- ils effectuent la distribution du flot de façon *géométriquement correcte*.

Algorithme 3 Minimal-Dense ($S = (O,C)$: un GCSP ; d : un entier) **retourne** S' : un GCSP)

Requiert: $d > 0$ représente la dimension de l'espace géométrique contenant S

Assure: $S' \subset S$ vérifie DDL(S')$< \frac{d(d+1)}{2}$ et $\forall S'' \subsetneq S'$, DDL($S''$)$> \frac{d(d+1)}{2}$, ou S' est vide

$S' \leftarrow$ `Dense(`S,d`)`

$Necessaires \leftarrow \emptyset$

$O' \leftarrow$ `Objets(`S'`)`

Pour chaque $o \in O' \setminus Necessaires$ **Faire**

$S'' \leftarrow$ `Sous-GCSP-Induit(`$O' \setminus \{o\}$,S'`)` {*S'' est le sous-GCSP de S' ne contenant pas l'objet o*}

$S'' \leftarrow$ `Dense(`S'',d`)`

Si $S'' \neq EmptyGCSP$ **Alors**

$S' \leftarrow S''$ {*S'' est dense, donc o n'est pas nécessaire; le sous-GCSP S' devient le nouveau sous-GCSP S'' identifié*}

Sinon

$Necessaires \leftarrow Necessaires \cup \{o\}$ {*o est nécessaire; il est conservé afin de ne pas être testé à nouveau*}

Fin Si

Fin Pour

Retourne S'

Ces deux améliorations sont obtenues au moyen de deux modifications majeures de la fonction `Distribute` dans la façon d'appliquer une surcharge dans le réseau objets-contraintes :

1. La valeur de la surcharge appliquée dans le réseau est le degré de rigidité au lieu d'être la constante $\frac{d(d+1)}{2}$.
2. La surcharge est appliquée via un nœud R dédié à cet effet, au lieu d'être appliquée sur chaque contrainte.

4.1 Nouvelle fonction `Distribute`

Comme nous l'avons vu dans la section précédente, le principe de la caractérisation structurelle de la rigidité est de fixer arbitrairement K déplacements et de vérifier l'existence d'un sous-GCSP bien-contraint. La fonction `Distribute` proposée par Hoffmann *et al.* applique ce principe en fixant les K déplacements via une contrainte, c-à-d. en fixant un repère local sur les objets géométriques liés à une seule contrainte. Cependant, retirer K déplacements à un sous-ensemble O' d'objets n'est géométriquement correct que si le sous-GCSP S' qu'il induit possède au moins K déplacements, c-à-d. qu'il vérifie DDR(S')$\geq K$.[4]

4. Rappelons que le DDR représente le nombre de déplacements admis par un sous-GCSP.

Considérons par exemple un segment en 3D, c-à-d. un sous-GCSP constitué de 2 points liés par une contrainte de distance. Ce sous-GCSP n'admet que 5 des 6 (3 rotations + 3 translations) déplacements autorisés par l'espace 3D puisque la rotation selon l'axe défini par le segment est sans effet sur celui-ci. Ainsi, retirer 6 déplacements à un segment 3D pour vérifier s'il est rigide est géométriquement incorrect et produit assurément un résultat erroné. C'est pourtant ce que fait la fonction `Distribute` proposée par Hoffmann *et al.* lorsqu'elle applique une surcharge $K = 6 + 1$ sur une contrainte de distance liant deux points dans un GCSP en 3D.

De façon à appliquer le principe de détection structurelle de rigidité de façon géométriquement correcte, nous proposons d'introduire une contrainte virtuelle R possédant K DDL. Cette contrainte, qui représente le fait de fixer K déplacements sur un ensemble d'objets, ne sera liée qu'aux sous-GCSP S' vérifiant DDR(S')$= K$. K et S' sont les deux paramètres d'entrée de notre version de la fonction `Distribute`.

Algorithme 4 Distribute (S : GCSP ; K : entier ; S' : GCSP) **retourne** S'' : GCSP)

Requiert: $K > 0$, $S' \subset S$ vérifie DDR(S')$= K$

Assure: $S'' \subset S$ vérifie DDL(S'')$< K$, ou S'' est vide

$G \leftarrow$ `Réseau-Surchargé`(S,K,S')

$V \leftarrow$ `Ford-Fulkerson`(G)

$S'' \leftarrow$ `Sous-GCSP-Induit`(V,S)

Retourne S''

La fonction `Réseau-Surchargé` est un peu différente de celle utilisée par Hoffmann *et al.*. Elle retourne le réseau objets-contraintes correspondant au GCSP S dans lequel la contrainte R a été introduite, ainsi que les arcs $s \rightarrow R$ et $R \rightarrow o$, pour chaque $o \in S'$, de capacités respectives K et $+\infty$. Le calcul du flot maximum dans ce réseau est toujours effectué par la fonction `Ford-Fulkerson`, qui retourne toujours l'ensemble V des nœuds-objets traversés lors de la dernière recherche d'un chemin augmentant, cet ensemble étant vide si le flot maximum sature tous les arcs issus de la source s. Selon la preuve de Hoffmann *et al.*, S'' vérifie DDL(S'')$< K$ ou S'' est vide.

Nos deux modifications de la fonction `Distribute` permettent d'une part d'appliquer la surcharge sur n'importe quel sous-GCSP, et d'autre part de ne distribuer la surcharge que vers les sous-GCSP pour lesquels cette opération est géométriquement correcte.

Exemple d'application de `Distribute`: Toujours sur le GCSP de la figure 1-a, notre fonction `Distribute` procède différemment de celle de Hoffmann *et al.*: puisque DDR(AB)=2, la surcharge maximale applicable sur ce sous-GCSP est $K = 2$. La figure 2-b présente l'appel `Distribute(`S,2,AB`)` pour la nouvelle version de la fonction. Le flot maximum parvient cette fois à saturer tous les arcs issus de la source et aucun sous-GCSP n'est donc retourné. Ceci est à la fois correct du point de vue structurel puisque DDL(AB) n'est pas strictement inférieur à $K = 2$, et du point de vue géométrique puisque ce sous-GCSP n'est pas sur-rigide.

Complexité de `Distribute:` La complexité de la nouvelle fonction `Distribute` est exactement la même que celle de la fonction initialement proposée par Hoffmann *et al.*. En effet, la seule différence réside dans la construction du réseau, et nos modifications se font en temps linéaire dans la taille du sous-GCSP à lier à la contrainte virtuelle R. Cette étape de construction est toujours dominée par l'étape de calcul du flot maximum, et la fonction demeure en $O(n * (n + m)^2)$, complexité en pire cas qui peut toujours être améliorée par les heuristiques disponibles pour la version originale de cette fonction.

4.2 Algorithmes pour la détection de rigidité

A partir de la fonction `Distribute`, Hoffmann *et al.* dérivaient les algorithmes `Dense` et `Minimal-Dense` qui permettent d'identifier des sous-GCSP rigides ou sur-rigides et de les minimiser (en nombre d'objets).

Ces algorithmes sont entièrement reproductibles avec la nouvelle fonction `Distribute`. Ceci nous permet de répondre aux mêmes problèmes mais de façon géométriquement correcte et avec une meilleure caractérisation de la rigidité : la rigidité structurelle étendue.

4.2.1 Détection de sous-GCSP sur-es_rigides

Schématiquement, l'algorithme `Dense` effectue des appels à la fonction `Distribute` pour chaque contrainte présente dans le GCSP considéré, et ce jusqu'à ce que cette fonction retourne un GCSP non-vide ou jusqu'à ce que toutes les contraintes aient été traitées. Dans cet algorithme, la surcharge est basée uniquement sur la dimension de l'espace géométrique considéré ; elle représente le nombre maximum de déplacements indépendants : 6 en 3D et 3 en 2D. L'algorithme `Dense` retourne donc des sous-GCSP sur-s_rigides[5] puisque les GCSP retournés n'admettent pas certains déplacements de l'espace géométrique considéré. Comme nous l'avons expliqué, cet algorithme est en réalité incorrect puisqu'il peut retirer parfois plus de déplacements que n'en admet le sous-GCSP lié à la contrainte surchargée.

Pour obtenir un algorithme correct, nous proposons d'utiliser la définition de la es_rigidité et notre fonction `Distribute` : la surcharge passée en paramètre à la fonction Distribute sera le DDR du sous-GCSP sur lequel on applique la surcharge.

Ceci nous permet de définir le nouvel algorithme `Over-Rigid` qui effectue un appel de la forme `Distribute(`S`,DDR(`S'`),`S'`)` pour chaque $S' \subset S$ jusqu'à ce qu'un sous-GCSP sur-es_rigide soit retourné ou que tous les S' aient été traités. En effet, un sous-GCSP S'' retourné par un appel de ce type vérifie nécessairement DDL(S'') $<$ DDR(S'), une condition suffisante pour que S'' soit sur-rigide (cf. définition 4 et lemme 1).

Bien sûr, un tel algorithme serait exponentiel en pire cas puisque le nombre de sous-GCSP dans un GCSP est égal au nombre de sous-ensembles d'objets dans l'ensemble d'objets de ce GCSP. Fort heureusement, nous montrerons (cf. section Propriétés de `Over-Rigid`) qu'il est suffisant d'appliquer la fonction `Distribute`

5. s_rigide ou sur-s_rigide si on utilise une surcharge augmentée de 1.

aux sous-GCSP *DDR-minimaux* uniquement (cf. définition 6 ci-après), ce qui produit l'algorithme suivant :

Algorithme 5 Over-Rigid (S : GCSP) retourne S'' : GCSP

Assure: $S'' \subset S$ est es_rigide ou sur-es_rigide ou vide

$S'' \leftarrow emptyGCSP$

$M \leftarrow$ `DDR-Minimaux(`S`)` {*construit l'ensemble des DDR-minimaux de* S}

Tant que $S'' = emptyGCSP$ et $M \neq \emptyset$ **Faire**

$S' \leftarrow$ `Pop(`M`)`

$S'' \leftarrow$ `Distribute(`S`,DDR(`S'`),`S'`)`

Fin Tant que

Retourne S''

Définition 6 GCSP DDR-minimaux
Un GCSP S est **DDR-minimal** *ssi il ne contient aucun sous-GCSP possédant le même DDR, c-à-d. que $\forall S' \subsetneq S$, DDR(S')<DDR(S).*

L'importance de cette modification vient du fait que la taille d'un sous-GCSP DDR-minimal est bornée en dimension donnée, et le nombre de tels sous-GCSP n'est donc pas exponentiel (cf. paragraphe ci-dessous sur la complexité de l'algorithme).

Exemple d'application de `Over-Rigid`

Considérons le GCSP S de la figure 1-b. Soit $M = \{BC,CEF,...\}$ l'ensemble de ses sous-GCSP DDR-minimaux produits par la fonction `DDR-Minimaux(`S`)`. L'algorithme `Over-Rigid` procède alors comme suit :

1. A la première itération, $S' = BC$ et K =DDR(BC)=5. La figure 2-c représente l'appel `Distribute(`S,5,BC`)`. Les arcs issus de la source s sont saturés par le calcul du flot maximum et aucun sous-GCSP sur-rigide n'est identifié.
2. A la seconde itération, $S' = CEF$ et K =DDR(CEF)=6. La figure 2-d présente l'appel `Distribute(`S,6,CEF`)` correspondant à cette itération. Cette fois-ci, on peut constater que l'arc $s \rightarrow R$ n'est pas saturé par le flot maximum. L'ensemble des objets atteignables à partir de la source dans le graphe résiduel étant $\{A,C,D,E,F\}$, l'algorithme termine à cette itération en retournant le sous-GCSP $S'' = ACDEF$ qui est effectivement sur-es_rigide.

Sur ce même GCSP, l'algorithme `Dense`, qui a le même objectif que notre algorithme `Over-Rigid` retournerait soit un segment (par exemple CD), soit un sous-GCSP composé de la droite et d'un point incident (par exemple AB), comme étant un sous-GCSP sur-rigide, ce qui est faux[6].

6. En pratique, l'algorithme `Dense` est assisté par des heuristiques qui lui permettent d'éviter des erreurs aussi triviales, mais il demeure géométriquement incorrect et peut toujours être trompé par des configurations non répertoriées dans l'heuristique [Jer02].

Propriétés de `Over-Rigid`

Afin de démontrer la correction et la complétude de l'algorithme, nous utiliserons les lemmes suivants qui établissent des propriétés du concept de DDR et de la distribution de flots dans des réseaux objets-contraintes :

Lemme 1 *Soit S un GCSP et $S' \subset S'' \subset S$ deux sous-GCSP. Alors DDR(S') $\leq$ DDR(S'').*

Démonstration : Chaque unité du DDR dans un GCSP représente un déplacement indépendant (translation or rotation) admis par ce GCSP. L'ajout d'un nouvel objet dans ce sous-GCSP ne peut en aucun cas retirer un déplacement de ce GCSP puisque les contraintes sont indépendantes du repère global. Ainsi, DDR(S') $\leq$ DDR($S' \cup \{o\}$).□

Lemme 2 *Soit $S' \subset S'' \subset S$ deux sous-GCSP du GCSP S. Si l'appel* `Distribute(`S,K,S''`)` *retourne un sous-GCSP S_0* non vide, *alors l'appel* `Distribute(`S,K,S'`)` *retourne nécessairement un sous-GCSP S_1* non vide.

Démonstration : Soit $G_{S'}$ le réseau surchargé utilisé par `Distribute(`S,K,S'`)` et $G_{S''}$ le réseau objets-contraintes surchargé utilisé par `Distribute(`S,K,S''`)`. La seule différence entre ces deux réseaux est la présence d'arcs supplémentaires du type $R \rightarrow o$ dans $G_{S''}$ puisque $S' \subset S''$. Ainsi, la distribution de flot est nécessairement plus *difficile* dans $G_{S'}$ que dans $G_{S''}$, les capacités totales issues de la source et entrant dans le puits étant identiques dans ces deux réseaux. Ainsi, si un calcul de flot maximum ne peut saturer les arcs issus de la source dans $G_{S''}$, il est impossible, a fortiori, qu'un flot maximum les sature dans $G_{S'}$.□

Correction de `Over-Rigid:`

Soit S'' un sous-GCSP *non vide* résultant de l'appel à `Over-Rigid`(S). Supposons que S'' résulte de l'appel `Distribute`(S,$DDR(S')$,S') pour un sous-GCSP S donné ; S'' vérifie alors nécessairement DDL(S'') $<$ DDR(S') puisqu'il est *non vide*. De plus, par définition de la fonction `Distribute`, $S' \subset S''$. Le lemme 1 assure alors que DDR(S')$\leq$DDR(S''), et on peut donc affirmer que DDL(S'')$<$DDR(S''), c-à-d. que S'' est sur-es_rigide.□

Complétude de `Over-Rigid:`

L'algorithme `Over-Rigid` applique la surcharge sur chaque sous-GCSP dans l'ensemble M des sous-GCSP *DDR-minimaux* (généré par la fonction `DDR-Minimaux(`S`)`). Remarquons que tout sous-GCSP non DDR-minimal contient par définition un sous-GCSP DDR-minimal ayant même DDR. Le lemme 2 assure qu'aucun sous-GCSP sur-es_rigide ne peut être manqué par l'algorithme.□

Complexité de `Over-Rigid:`

La complexité en temps de l'algorithme `Over-Rigid` dépend directement du nombre de sous-GCSP DDR-minimaux. Nous avons prouvé par énumération que la taille (en nombre d'objets) d'un sous-GCSP DDR-minimal est 2 et 2D et 3 en 3D pour

des GCSP constitués de points, droites et plans sous des contraintes de distances, incidences, angles et parallélismes[7]. Ainsi, le nombre de sous-GCSP DDR-minimaux dans un GCSP de ce type constitué de n objets en dimension d est en $O(n^d)$.

Appelons C_1 la complexité de la fonction `DDR-Minimaux`, et C_2 celle de la fonction `Distribute` que nous avons étudiée à la section précédente. Alors, la complexité en pire cas de l'algorithme `Over-Rigid` est $O(C_1 + n^d * C_2)$.

C_1 est généralement la complexité de la démonstration automatique en géométrie puisque le calcul du DDR nécessite la détermination des propriétés géométriques vérifiées par les objets du GCSP. C_1 est alors exponentielle en pire cas. Cependant, cette complexité peut être polynomiale voire constante pour des classes de GCSP particulières, comme les systèmes à barres ou les mécanismes génériques. Qui plus est, on peut employer plusieurs heuristiques dans le calcul du DDR qui permettent de rendre la complexité moyenne plus abordable en pratique. Dans ces cas là, C_1 est généralement négligeable en comparaison de C_2, et on obtient une complexité globale de l'algorithme `Over-Rigid` en $O(n^d * n * (n+m)^2)$. En comparaison, la complexité de l'algorithme `Dense` est en $O(m * n * (n+m)^2)$.

Notre algorithme induit donc un sur-coût approximativement linéaire en 2D et quadratique en 3D, puisque le nombre de contraintes m dans un GCSP est généralement $O(n)$ (moins d'équations que de variables dans un GCSP rigide).

4.2.2 Détection de sous-GCSP es_rigides ou sur-es_rigides

Afin d'identifier des sous-GCSP es_rigides ou sur-es_rigides, il suffit d'apporter une très légère modification à l'algorithme `Over-Rigid` présenté ci-dessus. Cette modification consiste simplement à ajouter 1 à la valeur de la surcharge appliquée à chaque appel de la nouvelle fonction `Distribute`. Celle-ci devient $K = DDR(S') + 1$ au lieu de $K = DDR(S')$. Il en résulte que les GCSP S'' non vides retournés par l'algorithme `Well-Or-Over-Rigid` vérifieront nécessairement $DDL(S'') \leq DDR(S')$, ce qui constitue une condition suffisante pour que S'' soit es_rigide ou sur-es_rigide (cf. définition 4 et lemme 1).

Toutes les propriétés de l'algorithme `Over-Rigid` sont préservées : l'algorithme `Well-Or-Over-Rigid` est correct, complet et de complexité $O(n^d * n * (n+m)^2))$.

Remarque : Même si le sous-GCSP S'' vérifie $DDL(S'') = DDR(S'')$, cela ne signifie pas qu'il est es_rigide. En effet, il faut encore vérifier la seconde condition de la es_rigidité, c-à-d. que S'' ne contient aucun sous-GCSP sur_es-rigide.

4.2.3 Déterminer la rigidité d'un GCSP

Pour décider si un GCSP S donné est es_rigide ou pas, il faut vérifier les deux conditions imposées par la définition de la es_rigidité (cf. définition 4):

- $DDL(S) = DDR(S)$
- S n'est pas sur-es_rigide, c-à-d. $\forall S' \subseteq S$, DDL(S')$\geq$ DDR(S')

7. La plupart des GCSP peuvent se ramener à des systèmes utilisant uniquement ces objets et contraintes primitives.

Pour vérifier la première condition, il suffit de calculer le nombre de DDL de S (par un simple compte, cf. définition 2) et son degré de rigidité DDR (voir [JNT02]).

Afin de vérifier la seconde condition, un simple appel à l'algorithme `Over-Rigid` suffit: si cet algorithme retourne un sous-GCSP vide, alors S n'est pas sur-es_rigide.

L'algorithme est complet, correct et de complexité égale soit à celle du calcul du DDR (si celle-ci est *cher*), soit égale à celle de l'algorithme `Over-Rigid` si le calcul du DDR est négligeable.

4.2.4 Minimisation d'un sous-GCSP es_rigides ou sur-es_rigide

Cet algorithme reprend le principe de minimisation proposé par Hoffmann *et al.* dans l'algorithme `Minimal-Dense`, qui visait à minimiser un sous-GCSP s_rigide ou sur-s_rigide. Il suffit de remplacer l'appel à `Dense` par un appel à `Well-Or--Over-Rigid`[8] dans l'algorithme `Minimal-Dense` présenté précédemment afin d'obtenir l'algorithme `Minimal-Rigid` qui retourne effectivement un sous-GCSP es_rigide ou sur-es_rigide minimal pour l'inclusion.

La correction et la complétude de cet algorithme sont encore une fois assurées par les propriétés des algorithmes `Minimal-Dense` et `Well-Or-Over-Rigid`. La complexité, quant à elle, est en pire cas $O(n * n^d * n * (n+m)^2))$ si le sous-GCSP initialement identifié est le GCSP entier et que tout objet doit effectivement être testé.

5 Conclusion

Dans cet article, nous avons proposé une nouvelle famille d'algorithmes permettant de répondre aux principales questions associées au concept géométrique de rigidité en nous appuyant sur une nouvelle caractérisation : la rigidité structurelle étendue. Ces algorithmes découlent des algorithmes à base de flots proposés par Hoffmann *et al.* pour la caractérisation par rigidité structurelle.

D'une part, nous avons montré dans [JNT02] que la caractérisation par rigidité structurelle étendue correspond strictement mieux à la rigidité que celle par rigidité structurelle, et d'autre part, nous venons de montrer que les algorithmes existants pour l'ancienne caractérisation pouvaient tous être reproduits pour la nouvelle caractérisation, moyennant un sur-coût approximativement linéaire en 2D et quadratique en 3D.

Nous pensons donc que toutes les méthodes d'analyse qualitative ou de résolution de GCSP actuellement basées sur la caractérisation par rigidité structurelle peuvent à présent considérer l'utilisation de notre nouvelle caractérisation. Cette alternative apportera vraisemblablement un gain significatif en fiabilité, mais également en généricité puisque les méthodes actuelles ont pour la plupart recours à des règles *ad-hoc* pour gérer les nombreuses exceptions connues à la rigidité structurelle classique. Il en résulte que ces méthodes pourront alors être utilisées dans des domaines d'application critiques où l'erreur est difficilement tolérable (par exemple, la conception aéronautique ou aérospatiale).

Le concept de degré de rigidité (DDR), qui s'avère central dans la définition de la nouvelle caractérisation, pose cependant encore quelques questions: en pire cas,

8. ou `Over-Rigid` si on souhaite seulement minimiser des sous-GCSP sur-es_rigides.

calculer le DDR d'un GCSP s'avère de l'ordre de la preuve formelle de théorème géométrique. Nous avons déjà identifié des classes de GCSP pour lesquelles ce problème est polynomial, mais un effort reste à faire sur ce sujet afin d'identifier d'autres classes d'une part, et de proposer des heuristiques de calcul permettant d'éviter le plus possible le recours à des algorithmes exponentiels d'autre part.

Par ailleurs, une comparaison expérimentale entre nos nouveaux algorithmes et les algorithmes proposés par Hoffmann *et al.* reste à mener. Ce travail permettra de juger en pratique du surcoût engendré par l'emploi de notre caractérisation au travers de nos nouveaux algorithmes. Toutefois, soulignons à nouveau que l'emploi de notre caractérisation et des algorithmes associés peut s'avérer malgré tout utile, voire indispensable, pour les applications où un gain en fiabilité important est plus critique qu'un gain en performance.

Références

[BFH+95] W. Bouma, I. Fudos, C.M. Hoffmann, J. Cai, and R. Paige. Geometric constraint solver. *Computer Aided Design*, 27(6):487–501, 1995.

[DMS98] J.-F. Dufourd, P. Mathis, and P. Schreck. Geometric construction by assembling solved subfigures. *Artificial Intelligence*, 99(1):73–119, 1998.

[FF62] L.R. Ford and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[HLS97] C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Finding solvable subsets of constraint graphs. In *Principles and Practice of Constraint Programming CP'97*, pages 463–477, 1997.

[HLS01] C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Decomposition plans for geometric constraint systems. In *Proc. J. Symbolic Computation 2000*, volume 39, pages 367–427, 2001.

[Jer02] C. Jermann. *Résolution de contraintes géométriques par rigidification récursive et propagation d'intervalles*. Thèse de doctorat, Université de Nice - Sophia Antipolis, 2002.

[JNT02] C. Jermann, B. Neveu, and G. Trombettoni. On the structural rigidity for gcsps. In *Proceedings of the 4^th International Workshop on Automated Deduction in Geometry*, 2002.

[JTNR00] C. Jermann, G. Trombettoni, B. Neveu, and M. Rueher. A constraint programming approach for solving rigid geometric systems. In *Principles and Practice of Constraint Programming, CP 2000*, volume 1894 of *LNCS*, pages 233–248, 2000.

[Kra92] G. Kramer. *Solving Geometric Constraint Systems*. MIT Press, 1992.

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

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