Résolution par contraintes de systèmes géométriques rigides

Christophe Jermann *
Université de Nice-Sophia Antipolis, I3S, ESSI, 930 route des Colles,
B.P. 145, 06903 Sophia Antipolis Cedex, email: jermann@unice.fr

Gilles Trombettoni
Université de Nice-Sophia Antipolis, I3S, ESSI, 930 route des Colles,
B.P. 145, 06903 Sophia Antipolis Cedex, email: trombe@essi.fr

Bertrand Neveu
CERMICS, 2004 route des lucioles,
B.P. 93, 06902 Sophia Antipolis Cedex, email: neveu@sophia.inria.fr

Michel Rueher
Université de Nice-Sophia Antipolis, I3S, ESSI, 930 route des Colles,
B.P. 145, 06903 Sophia Antipolis Cedex, email: rueher@essi.fr

Résumé

Cet article présente une nouvelle méthode de rigidification, utilisant la propagation d'intervalles, pour résoudre des systèmes de contraintes géométriques. Les techniques de rigidification sont des méthodes constructives basées sur le graphe de contraintes et exploitant les degrés de liberté des objets géométriques. Elles opèrent en deux phases : la phase de planification identifie les *clusters* rigides et la phase de résolution calcule les coordonnées des objets géométriques dans chaque cluster. Nous proposons une nouvelle heuristique dont l'objectif est de réduire la taille des sous-systèmes d'équations obtenus par planification. Nous montrons aussi que l'utilisation de la propagation d'intervalles permet non seulement l'implantation efficace de la phase de résolution mais généralise de plus les techniques ad-hoc classiques. De premiers résultats expérimentaux montrent que notre approche est plus efficace que des systèmes basés sur des techniques générales de décomposition.

1 Introduction

La modélisation par contraintes est une méthode attrayante dans le domaine de la CAO. Elle permet à l'utilisateur de construire un assemblage en spécifiant des contraintes

*Bourse de Thèse BDI en association avec le CNRS et la région Provence Alpes Côte d'Azur

géométriques de façon déclarative. En pratique, les systèmes géométriques contiennent de nombreuses parties rigides. La rigidification récursive (cf. [HLS97]) permet un calcul constructif des systèmes rigides en détectant et en assemblant des sous-parties rigides. Nous présentons un algorithme de rigidification suffisamment général pour prendre en compte des systèmes en 2 ou 3 dimensions. Cet algorithme produit une décomposition d'un système rigide en sous-systèmes rigides qui pourront être résolus séparément. Le but de cet article est de montrer que les techniques d'intervalles rendent cette approche sémantique efficace. De plus, lorsque les sous-systèmes obtenus sont suffisamment petits, la résolution peut être effectuée par des résolveurs symboliques.

L'article est organisé de la façon suivante : la section 1.1 présente le problème et la technique de rigidification récursive. La section 2 décrit la nouvelle étape de planification que nous proposons. La section 3 présente une étape de résolution basée sur la propagation d'intervalles. La section 4 présente quelques résultats expérimentaux.

1.1 Description du problème

Le problème que nous considérons est le calcul des positions et orientations possibles d'un ensemble *d'objets géométriques* satisfaisant un ensemble de *contraintes géométriques* qui les rendent rigides relativement les uns aux autres (cf. [FH97]).

Definition 1 *Un* **problème de contraintes géométriques** *est défini par un ensemble d'objets géométriques et un ensemble de contraintes géométriques.*

Un **objet géométrique** *est défini par un ensemble de paramètres dans un référentiel de dimension donné tel que le plan euclidien ou l'espace à 3 dimensions. Les* **paramètres** *définissent une position et une orientation pour un objet géométrique.*

Une **contrainte géométrique** *est une relation entre objets géométriques.*

Des exemples d'objets géométriques sont les points, les droites et les cercles en 2D, les points, les droites, les cylindres et les sphères en 3D. Les contraintes géométriques peuvent être l'incidence, la tangence, l'orthogonalité, le parallélisme, la distance ou l'angle.

Bien que l'algorithme présenté puisse prendre en compte un espace 3D, nous limiterons notre description aux éléments suivants du plan euclidien :

- Objets géométriques : points, droites et cercles.
- Contraintes géométriques : incidence, orthogonalité, parallélisme, distance et angle.

Nous supposerons aussi que :

- *Les contraintes géométriques sont binaires.* Ceci n'est pas une limite importante puisque la plupart des contraintes géométriques de base sont binaires. De plus, cette restriction ne porte pas sur la formulation algébrique des contraintes.
- *Tous les objets géométriques sont indéformables*, c'est-à-dire que les paramètres ne peuvent être indépendants d'un système de référence. De plus, les contraintes ne peuvent définir des relations qu'entre ces paramètres.

Par exemple, un cercle est défini par les deux coordonnées de son centre, mais son rayon doit être constant. Une contrainte de distance ne peut mettre en jeu une distance variable. Cette limitation est propre à la technique de rigidification qui utilise des déplacements (cf. [JASR99]).

1.2 Rigidification récursive

La rigidification récursive est basée sur une analyse des degrés de liberté du graphe pondéré de contraintes géométriques [HLS97].

Definition 2 *un* **graphe pondéré de contraintes géométriques** $G = (O,C)$ *est défini de la façon suivante :*

- *Un sommet de O représente un objet géométrique. Son poids est le nombre de* **degrés de liberté** *qu'il possède, c'est-à-dire, le nombre de coordonnées nécessaires pour le positionner de façon unique. Par exemple, un point a 2 degrés de liberté dans le plan Euclidien.*
- *Une arête de C représente une contrainte géométrique. Son poids $w(C)$ est le nombre de paramètres qui sont fixés par la contrainte ; généralement, cela correspond au nombre d'équations de la forme algébrique de la contrainte.*

L'analyse des degrés de liberté exploite une propriété structurelle du graphe pondéré de contraintes géométriques : la *rigidité structurelle*[1].

Definition 3 *Soit S un problème de contraintes géométriques, et $G = (O,C)$ le graphe pondéré de contraintes géométriques correspondant. Soit W la fonction qui calcule la différence entre le poids des objets et celui des contraintes :*
$W(G) = \sum_{o \in O}(w(o)) - \sum_{c \in C}(w(c))$.
En dimension d, le système S est **structurellement rigide (s-rigide)** *ssi :*

- $W(G) = d(d+1)/2$
- *Pour tout sous-graphe G' de G, $W(G') \geq d(d+1)/2$*

Un sous-graphe s-rigide sera appelé un **cluster**. *Ainsi, en 2D, un cluster possède 3 degrés de liberté qui lui permettent de se déplacer et de tourner dans le plan.*

La s-rigidité est similaire à la propriété P énoncée dans [LM98] et à la notion de *densité* définie dans [HLS97].

La rigidification récursive produit de façon itérative de nouveaux clusters dans le graphe : un seul sommet remplace les objets du cluster, les arêtes connectant des objets du cluster à un objet hors du cluster sont combinées en une seule arête pondérée par la somme de poids des arêtes combinées. La figure 1 illustre ces notions.

La phase de planification a pour but la décomposition du système complet en une séquence de petits sous-systèmes. Elle entremêle deux étapes : fusion et extension :

- L'**étape de fusion** détecte dans le graphe un petit cluster formé de plusieurs objets et clusters.
- L'**étape d'extension** étend le cluster obtenu en ajoutant les objets voisins un par un. Chaque addition désignera *une* extension par la suite.

Il apparaît que l'étape de fusion est suffisante pour la construction du plan. Cependant, cette étape peut avoir à parcourir le graphe dans son ensemble. Aussi, l'étape d'extension doit être vue plutôt comme une heuristique permettant une fusion *efficace* : la s-rigidité est vérifiée incrémentalement à chaque ajout d'objet dans le cluster courant.

1. la s-rigidité est une condition nécessaire à la rigidité, mais non suffisante sauf dans le cas de contraintes de distance entre points en 2D. Elle est cependant communément accepté en tant que bonne heuristique pour la détection de sous-parties rigides d'un système de contraintes géométriques. Plusieurs contre-exemples en 2 et 3 dimensions montrent que des contraintes redondantes sont la principale cause d'échec de la détection de sous-parties rigides à l'aide de la s-rigidité [Hen92].

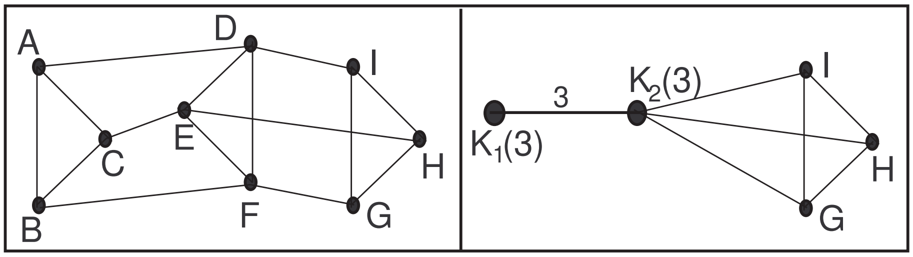

FIG. 1 – *Graphes associés à un problème de contraintes géométriques en 2D. (A gauche) Tous les objets sont des points (poids 2), les arêtes représentant des contraintes de distance entre eux (poids 1). Toute paire de points connectés forme un cluster. Tout triangle est aussi un cluster. Les points B, C et E ne forment pas un sous-graphe s-rigide car il possède encore 4 degrés de liberté. (A droite) Un autre graphe où les triangles (A,B,C) et (D,E,F) ont été condensés en 2 clusters K_1 et K_2. l'arête de poids 3 représente la combinaison des arêtes (A,D), (C,E) et (B,F) du graphe précédent.*

La phase de résolution, aussi appelée phase de construction, suit le plan donné par la phase précédente et calcule les coordonnées de chaque objet dans chaque cluster.

1.3 Travaux existants

La technique de rigidification récursive a été développée [VSR92, BFH$^+$95, FH93, FH97, DMS98] pour assembler des points et des droites sous des contraintes de distance et d'angles en 2D. [HV95] et [Kra92] décrivent une première tentative de travail en 3D. Tous ces systèmes utilisent des algorithmes spécifiques pour fusionner deux ou trois clusters prédéfinis. Les modèles de constructions possibles sont définis dans une bibliothèque.

Hoffmann et al. [HLS97, HLS98] ont introduit une méthode basée sur un algorithme de flot pour le calcul de l'étape de fusion. Cet algorithme trouve un *sous-graphe dense minimal* du graphe pondéré de contraintes géométriques, c'est-à-dire qu'il calcule un cluster s-rigide de taille minimale (il n'existe pas de sous-partie de ce cluster qui soit elle-même s-rigide). Des méthodes de résolution ad-hoc sont utilisées pour la phase de construction. Cet algorithme peut travailler dans un espace de n'importe quelle dimension (en particulier 2D et 3D), et peut être appliqué sur n'importe quel type d'objets géométriques.

La principale limite de l'approche Hoffmann vient du fait qu'aucune méthode n'est proposée pour la phase de résolution. Les méthodes de résolution symbolique pourraient être utilisées mais ne sont en général pas efficaces pour ce type de problème. Les méthodes spécifiques employées par les anciens algorithmes de rigidification pourraient aussi être utilisées, mais il faudrait alors qu'elles soient définies pour tout cluster que l'algorithme de flot est capable de générer.

1.4 Contribution

Dans cet article, nous proposons :

1. Une nouvelle heuristique pour l'étape d'extension de l'algorithme de planification de Hoffmann [HLS97]. L'objectif est d'obtenir de plus petits sous-systèmes d'équations.

2. Une nouvelle méthode de résolution plus générale basée sur les techniques d'intervalles. Les algorithmes de réduction d'intervalles [Lho93, HMD97] opèrent sur des intervalles de flottants et peuvent approximer les solutions d'un système d'équations numériques par filtrage et dichotomie. Dans nos travaux, ils sont utilisés pour calculer les solutions de tous les sous-systèmes rigides. L'utilisation des techniques d'intervalles pour la phase de résolution a deux avantages. D'abord cette approche est générale et peut remplacer les méthodes spécifiques définies pour chaque modèle de construction. Ensuite, aucune solution n'est perdue.

2 La phase de planification

Le but de cette phase est de trouver un ordre permettant de résoudre les contraintes de façon incrémentale. Plus précisément, il s'agit d'identifier les sous-parties rigides qui peuvent être résolues indépendemment (puis assemblées).

Hoffmann et al. [HLS97] ont présenté un algorithme accomplissant une telle phase de planification. Sa principale limite vient du fait qu'il requiert parfois l'ajout de gros blocs de contraintes qui peuvent rendre la résolution coûteuse. Nous proposons ici une heuristique dont le but est de réduire le nombre de contraintes ajoutées à chaque étape.

Les paragraphes suivants illustrent le principe et les limites de l'algorithme de planification proposé par Hoffmann sur un court exemple. Nous détaillons ensuite l'heuristique proposée.

2.1 L'algorithme de Hoffmann et al.

L'algorithme de planification proposé par Hoffmann et al. construit un arbre renversé de clusters appelé *arbre de clusters*: la racine est le cluster final qui contient tout le système ; les feuilles sont les objets géométriques ; il existe un arc entre un cluster C et tous les clusters qui sont fusionnés pour donner C.

L'algorithme de Hoffmann construit des clusters en entrelaçant des étapes de fusion et d'extension. Il termine lorsque l'étape de fusion échoue. Cet échec intervient lorsque le système a été entièrement rigidifié, ou lorsque le système complet n'est pas s-rigide. L'algorithme met à jour un graphe pondéré de contraintes géométriques G_m à chaque étape de fusion et d'extension. Considérons par exemple un système en 2D constitué de 15 points et de 27 contraintes de distance les reliant (voir figure 2, G_m^0).

La première étape de fusion trouve le sous-graphe $K_1 =< A,B >$ de G_m. Ce cluster est étendu jusqu'à obtention d'un point fixe. L'ensemble des objets adjacents à ce cluster est $\{C,H,I\}$. $< A,B,I >$ étant s-rigide, I est ajouté à K_1. H, C et J sont ajoutés à K_1 de la même façon. L'ensemble des objets adjacents à K_1 est maintenant $\{D,G,N,K\}$. Aucun de ces points ne peut être ajouté par extension, nous avons donc atteint le point fixe $K_1 =< A,B,I,H,C,J >$. Un nouveau sommet K_1 de poids 3 est ajouté à G_m et les sommets A,B,I,H,C,J sont retirés du graphe (ainsi que toutes les contraintes les reliant, cf figure 2 G_m^1). K_1 est ajouté à l'arbre de clusters et l'étape de fusion suivante commence. Elle identifie $< M,N >$ qui est ensuite étendu en $K_2 =< M,N,O,K,L >$. Notons que K_2 aurait pu être étendu sur J si $< M,N >$ avait été détecté en premier. Finalement, G,F,D et E sont agrégés dans K_3 (C et L auraient aussi pu l'être si K_3 avait été construit en premier).

Les clusters ne partageant pas de points, les contraintes inter clusters (contraintes de distance dist(J,K), dist(J,N), dist(G,C), dist(G,L), dist(F,L) et dist(C,D)) sont incluses

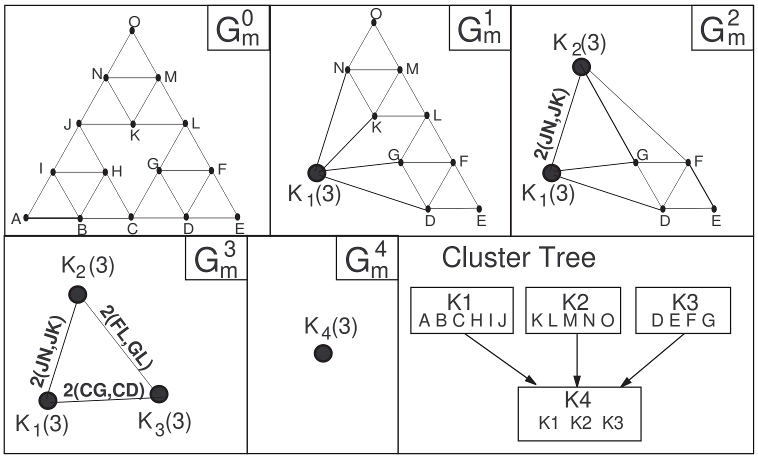

FIG. 2 – *Mises à jour de G_m lors du déroulement de la planification de Hoffmann, et arbre de clusters résultant.*

uniquement lors de la dernière étape de fusion qui risque de rendre la résolution correspondante relativement coûteuse.

C'est pourquoi nous proposons une nouvelle heuristique permettant aux clusters de partager des objets géométriques, le but étant de maximiser le nombre d'étapes d'extension, et de réduire ainsi le nombre de contraintes mises en jeu lors des étapes de fusion. Cette heuristique généralise les anciennes méthodes ad-hoc [BFH$^+$95].

2.2 Heuristique proposée

Tout comme l'algorithme de Hoffmann, l'algorithme `Rigidification` entrelace étapes de fusion et d'extension. Pour faciliter le partage d'objets, il utilise deux graphes :

- le graphe de fusion G_m semblable à celui utilisé par l'algorithme de Hoffmann. Ce graphe est utilisé uniquement pour l'étape de fusion.
- le graphe d'extension G_e spécialement utilisé pour les étapes d'extension. Il contient uniquement des objets géométriques.

L'algorithme accomplit trois étapes principales itérativement (boucle `while`) : étape de fusion, étape d'extension et étape de mise à jour.

Une étape de fusion est accomplie au moyen de la fonction `MinimalSRigid` (G_m, d, G_e). Cette fonction calcule G_1, un sous-graphe dense minimal de G_m, au moyen de l'algorithme de flot décrit dans [HLS97]. Puis ce sous-graphe est traduit en un sous-graphe équivalent dans G_e et renvoyé comme résultat. L'ensemble vide est renvoyé si G_m contient un seul sommet ou si aucun sous-graphe de G_m n'est s-rigide.

Une étape d'extension étend le cluster G_1 renvoyé par l'étape de fusion. La boucle `repeat` tente incrémentalement d'ajouter un objet à G_1. Un objet est ajouté si le cluster résultant reste s-rigide.

La dernière étape met à jour les graphes G_m et G_e ainsi que l'arbre de clusters.

Mise à jour de G_m

Un nouveau cluster C est créé dans G_m et remplace les clusters et objets qu'il agrège (sous-graphe G_2). Cette opération est effectuée par la fonction `Condense`(G_2, C, G_m) de la façon suivante : (a) remplacer les sommets de G_2, sous-graphe de G_m correspondant à G_1, par un sommet unique C dans G_m ; (b) combiner toutes les arêtes entre un sommet v de $G_m - G_2$ et des sommets de G_2 en une arête unique entre v et C : son poids est la somme des poids des arêtes combinées.

Le nouveau cluster C peut contenir des objets qui ont déjà été agrégés auparavant dans d'autres clusters (c.a.d des objets partagés). Des arêtes de coïncidence sont alors ajoutées dans G_m pour prendre ces partages en compte. Intuitivement, ces contraintes de coïncidence sont ajoutées pour préserver le bon nombre de degrés de liberté dans G_m. Elles explicitent le fait que les différentes occurrences d'un objet partagé représentent toutes le même objet (cf. fonction `AddCoincidences`(C, G_m)).

Algorithm 1 `Rigidification` (**in** G : Graphe ; **in** d : Entier ; **out** CT : ArbreDeClusters)

{G est le graphe pondéré de contraintes géométriques initial ; d est la dimension du problème (2D, 3D) ; CT est le plan (arbre de clusters), résultat de l'algorithme.}
$CT \leftarrow \emptyset$; $G_m \leftarrow G$; $G_e \leftarrow G$
$G_1 \leftarrow$ `MinimalSRigid`(G_m, d, G_e) *{Première étape de fusion}*
while $G_1 \neq \emptyset$ **do**
  {Etape d'extension}
  repeat
    for all $o \in$ `ConnectedVertices`(G_e, G_1) **do** {o est un sommet de G_e relié à au moins un objet de G_1}
      if $G_1 \cup \{o\}$ est s-rigide **then**
        {Ajoute o ainsi que les arêtes correspondantes à G_1}
        `AddVertex`(G_1, o, G_e)
      end if
    end for
  until Point fixe *{G_1 n'est plus modifié par extension}*
  {Etape de mise à jour}
  $G_2 \leftarrow$ `Convert`(G_1, G_m) *{G_2 est un sous-graphe deG_m correspondant à G_1}*
  `Condense`(G_2, C, G_m) *{Remplace G_2 par un nouveau sommet C dans G_m}*
  `AddCoincidences`(C, G_m)
  `InsertCluster`(C, CT) *{Insère C dans l'arbre de clusters}*
  `Rigidify(InterfaceObjects`(G_1)`)` *{Ajoute les contraintes interfaces de G_1 dans G_e}*
  `RemoveVertices(InternalObjects`(G_1)`,` G_e`)` *{Retire les objets internes de G_e ainsi que toutes les arêtes les reliant}*
  $G_1 \leftarrow$ `MinimalSRigid`(G_m, d, G_e) *{Etape de fusion}*
end while

Mise à jour de G_e

Les sommets du nouveau cluster créé C sont partitionnés en deux ensembles : les *objets interfaces* qui sont connectés à des objets de $G_e - G_1$ et les *objets internes*. La fonction `RemoveVertices` retire les objets internes car ils sont s-rigides relativement les uns aux autres (cela vient du fait qu'ils appartiennent au même cluster C). Les objets interfaces, eux, restent dans G_e car ils peuvent encore être partagés par de futurs clusters.

Pour maintenir le bon nombre de degrés de liberté, les objets interfaces restant dans G_e doivent être rigidifiés relativement les uns aux autres à l'aide de *contraintes interfaces*. La fonction `Rigidify` ajoute des contraintes interfaces entre ces objets de la façon suivante : si C ne contient que deux objets interfaces o_1 et o_2, une arête pondérée (o_1,o_2) est ajoutée dans G_e pour les rigidifier. S'il y a plus de deux objets interfaces, tout autre objet interface o_i de C est rigidifié par l'ajout des arêtes pondérées (o_i, o_1) et (o_i, o_2) dans G_e (cf. [JTNR00]).

L'algorithme *Rigidification* termine car le nombre d'objets dans G_m décroît à chaque étape. Sa correction est assurée par le fait que la propriété de s-rigidité est maintenue dans G_m et G_e tout au long de son exécution.

2.3 Exemple

La figure 3 illustre le comportement de l'algorithme `Rigidification` sur l'exemple présenté à la figure 2.

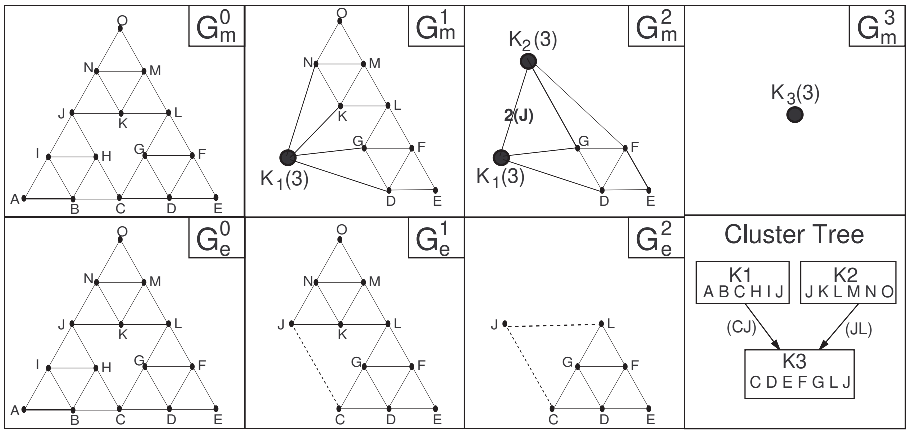

FIG. 3 – *Evolution des graphes G_m et G_e au cours de l'exécution de l'algorithme 1, et arbre de clusters obtenu. Les contraintes interfaces sont tracées en pointillés.*

Les premières étapes de fusion et d'extension sont similaires à celles produites par l'algorithme de Hoffmann et conduisent à G_1 qui contient A, B, I, H, C, J. Un nouveau sommet de poids 3, K_1, est ajouté à G_m^0. A, B, H, I, objets internes de ce nouveau cluster, sont retirés de G_e^0, ainsi que les arêtes internes. Une contrainte interface de poids 1 reliant J et C est ajoutée à G_e^0.

Le cluster K_2 est ensuite créé de la même façon. Il est condensé dans G_m^1. Comme K_2 contient le point J qui appartient déjà à K_1, une contrainte de coïncidence de poids 2 est ajoutée entre K_1 et K_2 dans G_m^1. Une contrainte interface JL est ajoutée dans G_e^1

Le cluster K_3 est finalement créé, incluant tous les points restants de G_e^2 et utilisant pour cela les contraintes interfaces. La phase de planification est alors terminée (G_m^3 contient un seul sommet).

2.4 Comparaison avec l'approche de Hoffmann

L'algorithme de Hoffmann utilise le même graphe pour les étapes de fusion et d'extension alors que l'algorithme `Rigidification` accomplit les étapes d'extension sur un graphe spécifique contenant les objets partagés. Ainsi, l'algorithme `Rigidification` peut être capable d'accomplir plus d'extensions. Il est important de comprendre qu'une extension génère un sous-système contenant au plus 3 équations en 2D (6 en 3D), c'est-à-dire le nombre de degrés de liberté de l'objet agrégé. Puisque l'algorithme maximise le nombre d'extensions, il devrait réduire le nombre d'étapes de fusion.

Par exemple, sur le système précédent, l'algorithme de Hoffmann construit les clusters K_1, K_2 et K_3 avant de les fusionner en K_4 (voir figure 2). Cela correspond à 9 extensions et 4 fusions, la dernière devant fusionner 3 clusters avec 6 contraintes de distance entre eux. Sur ce même exemple, l'algorithme 1 accomplit 13 extensions mais seulement 3 fusions, chacune de ces opérations ne mettant pas plus de 2 contraintes en jeu.

3 La phase de résolution

Cette section présente l'utilisation des techniques d'intervalles pour la résolution du plan produit par la phase de planification.

Les étapes atomiques de la phase de planification génèrent des sous-systèmes d'équations, appelés *blocs*, qui peuvent être résolus en séquence. Les techniques d'intervalles sont utilisées pour résoudre tous les blocs et conduisent à des solutions numériques[2]. Lorsqu'une solution est trouvée pour un bloc, les variables correspondantes sont remplacées dans les blocs suivants par leurs valeurs. Lorsque la résolution d'un bloc échoue, un retour arrière intervient et une autre solution est recherchée pour un bloc précédent. Cette partie détaille la génération des blocs à partir du plan et présente différentes méthodes pour résoudre le système décomposé.

3.1 Génération des équations

Un graphe orienté sans circuit (DAG) de blocs (cf. figure 4) est créé lors de la génération du plan.

- un **bloc** contient un (sous-)système d'équations. Ces équations correspondent aux arêtes (contraintes géométriques) retirées de G_e durant une étape de fusion ou une extension ;
- Il existe un arc d'un bloc A vers un bloc B si une variable dont la valeur est calculée dans A apparaît aussi dans une équation de B.

Montrons maintenant comment les contraintes interfaces sont traduites en équations. Chaque bloc résout son propre ensemble de variables. Les variables correspondant à des objets interfaces sont dupliquées pour chaque cluster dans lesquels elles apparaissent.

2. On calcule en fait un sur-ensemble des solutions. En effet, les parties de l'espace de recherche éliminées ne contiennent jamais de solutions, cependant les petits intervalles restants peuvent ne pas contenir de solutions.

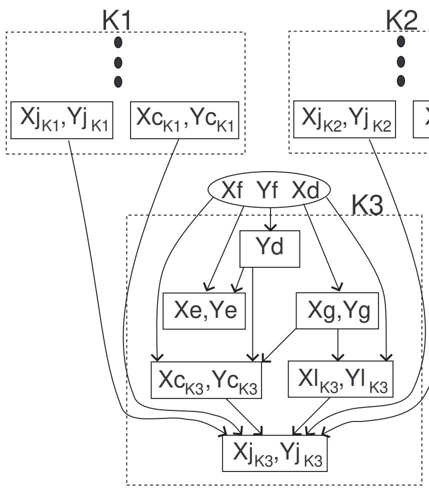

FIG. 4 – *DAG de blocs associé à l'arbre de clusters de la figure 3. Les blocs sont représentés par les petits rectangles contenant la liste des variables qu'ils calculent. Tous les blocs du cluster K_3 sont représentés. Le bloc sommet de K_3 est créé par l'étape de fusion, les blocs descendants correspondant aux extensions successives. Le dernier bloc de K_3 contient les deux contraintes interfaces ajoutées durant la création de K_1 et K_2.*

L'objet J est partagé par les clusters K_1, K_2 et K_3 et génère donc les variables $x_{J_{K_1}}$, $y_{J_{K_1}}$, $x_{J_{K_2}}$, $y_{J_{K_2}}$, $x_{J_{K_3}}$, $y_{J_{K_3}}$. Les variables $x_{J_{K_1}}$, $y_{J_{K_1}}$ (resp. $x_{J_{K_2}}$, $y_{J_{K_2}}$) sont calculées lors de la résolution des blocs du cluster K_1 (resp. K_2). Lors de la résolution du cluster K_3, le bloc calculant $x_{J_{K_3}}$, $y_{J_{K_3}}$ contient les deux contraintes interfaces $\text{dist}(J_3,L_3) = \text{dist}(J_2,L_2)$ et $\text{dist}(J_3,C_3) = \text{dist}(J_1,C_1)$. Ces équations utilisent les valeurs calculées pour J et L dans K_1 et K_2.

Nous décrivons maintenant les différentes méthodes de résolution basées sur les techniques de réduction d'intervalles que nous proposons.

3.2 Dichotomie et filtrage par bloc

Une première approche pour résoudre le DAG de blocs a été décrite au début de la section. Un filtrage standard associé à la dichotomie permet le calcul de l'ensemble des solutions de chaque bloc, et une étape de retour arrière entre blocs intervient lorsque le bloc courant n'a pas de solution.

Effectuer la réduction d'intervalles sur un bloc est simple : toute variable du bloc est soumise au filtrage et à la dichotomie. Il faut cependant être attentif au processus inter blocs : les valeurs calculées, qui remplaceront les variables dans les blocs suivants, ne sont pas des flottants mais des intervalles de flottants (même très petits, par exemple de taille 10^{-8}). Nous pourrions prendre en compte de tels intervalles en modifiant légèrement la fonction LNAR de Numerica [HMD97][3].

Nous préférons cependant une méthode plus simple : le point milieu de l'intervalle constant réduit remplace la variable pour les blocs suivants. Cette heuristique de point

3. La fonction LNAR, appliquée à une variable dans une équation, remplace toutes les autres variables par des intervalles constants et recherche la solution la plus à gauche.

milieu est simple et générale. Elle est de plus correcte si l'ensemble des intervalles obtenus à la fin est vérifié par une passe de filtrage sur toutes les équations du système. En pratique, cette vérification finale est très rapide car les intervalles sont petits. Bien sûr cette technique ne garantit pas l'obtention de toutes les solutions, mais en pratique aucune solution n'a été perdue sur les exemples traités.

Cette méthode est très efficace car remplacer des variables par des constantes équivaut à simplifier symboliquement les équations.

3.3 Propagation sur le système complet

Une autre méthode consiste à limiter la dichotomie au bloc courant mais à effectuer le filtrage sur le système complet. Elle peut être réalisé deux façons différentes :

1. Tous les blocs sont gérés par une technique de retour arrière inter blocs, comme pour la méthode décrite dans le paragraphe précédent. L'algorithme doit alors gérer deux systèmes : un système contenant les équations du bloc courant dans lequel filtrage et dichotomie sont appliqués comme précédemment, et un second système contenant les équations de l'ensemble des blocs non encore résolus dans lequel un filtrage intervient lorsqu'une réduction de domaine a eu lieu dans le premier système. Nous appellerons cette approche *résolution par bloc avec propagation* dans la suite de l'article.
2. Une autre approche, où toutes les équations sont dans un système unique, est appelée *résolution globale*. Elle consiste à considérer le DAG de blocs obtenu comme base d'une heuristique de choix de variable : seules les variables du bloc courant sont soumises à la dichotomie jusqu'à ce que tous les intervalles correspondants soient suffisament petits. Le filtrage, lui, s'effectue sur le système complet.

La résolution globale est plus simple à mettre en œuvre que la résolution par bloc avec propagation. Cependant, elle est moins efficace car elle ne peut pas bénéficier de l'heuristique de point milieu.

Conceptuellement, il est possible de mettre en œuvre les mécanismes de résolution par bloc avec tout résolveur permettant l'obtention de toutes les solutions d'un bloc. Les méthodes symboliques peuvent être utilisées lorsque les équations ne contiennent pas de fonction trigonométrique. L'utilisation d'un tel résolveur avec notre processus inter bloc permet d'assurer la complétude de la résolution et peut être envisagée pour des petits blocs. A l'inverse, les méthodes numériques classiques ne sont pas envisageables car elles ne permettent pas d'obtenir plusieurs solutions ; et même pour l'obtention d'une seule solution, les solutions obtenues pour chaque bloc pourraient ne pas être combinables en une solution pour le système complet.

3.4 Unification des systèmes de coordonnées

Une fois la phase de résolution terminée, chaque objet du cluster racine est placé dans le système de coordonnées final. Seuls les objets internes des autres clusters doivent encore subir un déplacement pour être placés dans ce système. Pour ce faire, l'arbre de clusters est parcouru depuis la racine et, à chaque noeud traversé, on applique une transformation. Plus précisément :

1. Les coordonnées des objets interfaces sont connues dans le système de coordonnées (car l'arbre de clusters est parcouru depuis la racine) et sont donc utilisées pour calculer les coefficients de la transformation.

2. La transformation ainsi obtenue est appliquée aux coordonnées de tous les objets internes du clusters, donnant leurs coordonnées dans le système de coordonnées final.

Comme chaque objet ne devient objet interne qu'une seule fois (lorsqu'il est agrégé dans un cluster et ne possède plus de contraintes vers des objets extérieurs), le parcours depuis la racine de l'arbre de clusters permet de n'appliquer la transformation qu'une seule fois à chaque objet.

4 Résultats expérimentaux

Cette section présente des résultats préliminaires sur trois exemples (voir figure 5). Ces exemples sont constitués de points soumis à des contraintes de distance en 2D. Les valeurs des distances ont été réglées de façon à ce que le nombre de solutions soit peu important, ceci afin de pouvoir comparer les temps de résolution des différentes méthodes présentées (Ex1 : 128 solutions ; Ex2 : 64 solutions ; Ex3 : 256 solutions)

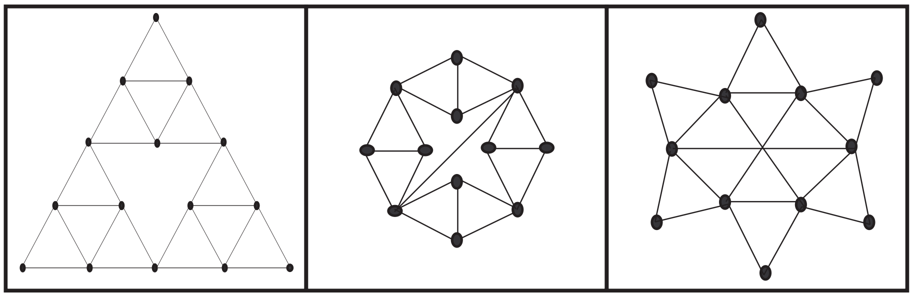

FIG. 5 – *De gauche à droite, les trois exemples en 2D, constitués de points et de contraintes de distances : Triangles (Ex1), Losanges (Ex2), et Hexagone (Ex3).*

Nous comparons tout d'abord la technique de décomposition par rigidification récursive à une approche équationnelle [AAJM93, BNT98] travaillant au niveau des relations équations-variables. Cette approche est basée sur une analyse structurelle du graphe de contraintes biparti équations-variables qui utilise le couplage maximum et la décomposition de Dulmage et Mendelsohn. Nous appliquons les mêmes techniques de résolution à cette décomposition. Pour des raisons de simplicité, nous adopterons les abréviations suivantes dans la suite de l'article :

- ED pour la décomposition équationnelle basée sur le couplage maximum ;
- SD1 pour la rigidification récursive avec partage d'objets introduite dans cet article (section 2 - Algorithme `Rigidification`) ;
- SD2 pour l'algorithme de rigidification récursive proposé par Hoffmann et al. .

4.1 Taille du plus gros bloc

Le tableau 1 montre que SD1 décompose les trois exemples en plus petits blocs que SD2 et ED.

Les deux premiers exemples sont décomposés par SD1 en petits blocs de deux équations au plus. Avec SD2, il reste un bloc de taille 10 (resp. 12) qui peut encore être décomposé par ED et donne un bloc de taille 6 (resp. 6). ED ne peut en revanche pas

exemples	SD1	SD1+ED	SD2	SD2+ED	ED	ND
Ex1	2	2	10	6	14	26
Ex2	2	2	12	6	10	20
Ex3	8	8	8	8	8	20

TAB. 1 – *Taille du plus gros bloc obtenu par rigidification avec objets partagés (SD1) et sans objets partagés (SD2), par rigidification avec et sans objets partagés et suivie d'une décomposition équationnelle (SD1+ED, SD2+ED), par la décomposition équationnelle seule (ED). La dernière colonne (ND) indique la taille du système complet (non décomposé).*

décomposer plus finement les résultats de SD1. Le troisième exemple présente un bloc de taille 8 dans les deux cas (SD1, SD2) qui ne peut être décomposé plus finement par ED.

4.2 Résolution par réduction d'intervalles

Nous avons résolu les trois exemples pour les décompositions SD1+ED, SD2+ED et ED. SD1+ED est équivalente à SD1 seule (cf. paragraphe 4.1), SD2+ED donne en revanche pour le même système une décomposition en plus petits blocs que SD2 seule. Notre implantation ne nous permet pas de fournir les temps pour SD2 seule, mais ceux-ci sont nécessairement moins bons que ceux de SD2+ED. Nous fournissons aussi les temps pour la résolution sans décomposition (ND). Pour les systèmes décomposés, nous appliquons les trois méthodes présentées dans les sections 3.2 et 3.3 : la résolution par bloc (M1), la résolution par bloc avec propagation (M2) et la résolution globale (M3).

Toutes les expérimentations ont été effectuées sur un Pentium III 500, et utilisent la bibliothèque IlcNumerica d'Ilog Solver [ILO98] qui met en œuvre le filtrage *Box-Consistency* [BAH94].

Exemples	**SD1+ED**			**SD2+ED**			**ED**			**ND**
	M1	M2	M3	M1	M2	M3	M1	M2	M3	-
Ex1	17	9	455	43	28	1322	58	29	385	5795
Ex2	1.4	11	77	9	13	178	56	117	467	6640
Ex3	0.9	3.4	289	2.6	12.2	1646	1.5	3.7	533	2744

TAB. 2 – *Temps CPU (en secondes) pour la résolution des décompositions SD1, SD1+ED, SD2+ED et ED par les trois méthodes M1, M2 et M3, et résolution sans décomposition (ND).*

4.3 Analyse

Les résultats montrent que :

- Décomposer est toujours fructueux : sans décomposition, le temps de résolution peut être plus important de 2 ordres de grandeur.
- La décomposition sémantique (SD1) basée sur la rigidification conduit en général à des blocs plus petits que la décomposition équationnelle (ED). Malgré la duplication

des objets partagés, le gain en temps reste significatif grâce à la réduction de la taille du plus gros bloc, ce qui explique aussi le gain par rapport à SD2.
– Les méthodes M1 et M2 donnent de meilleurs résultats que la méthode M3, ce qui montre l'intérêt de l'heuristique du point milieu.
– L'effet de la propagation dépend du problème : lorsque de nombreux retours arrières interviennent, comme pour Ex1, la propagation globale (M2) est utile.

5 Conclusion

Cet article a présenté une méthode complète pour traiter les problèmes de contraintes géométriques. Elle se compose :

1. d'une nouvelle heuristique pour la technique de rigidification récursive conduisant à des systèmes d'équations plus petits. Cette décomposition sémantique conduit à des sous-systèmes plus petits que les décompositions équationnelles.
2. d'une phase de résolution basée sur les techniques d'intervalles. Cette approche est générale et ne perd pas de solution. C'est une alternative prometteuse aux techniques symboliques ou numériques classiques.

De plus amples expérimentations restent à mener pour valider cette méthode.

Références

[AAJM93] Samy Ait-Aoudia, Roland Jegou, and Dominique Michelucci. Reduction of constraint systems. In *Compugraphic*, 1993.

[BAH94] F. Benhamou, D. Mc Allester, and P. Van Hentenryck. Clp(intervals) revisited. In *Proc. Logic Programming, MIT Press*, 1994.

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

[BNT98] Christian Bliek, Bertrand Neveu, and Gilles Trombettoni. Using graph decomposition for solving continuous csps. In *Principles and Practice of Constraint Programming, CP'98*, volume 1520 of *LNCS*, pages 102–116. Springer, 1998.

[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.

[HMD97] Pascal Van Hentenryck, Laurent Michel, and Yves Deville. *Numerica: A Modeling Language for Global Optimization.* MIT Press, 1997.

[HV95] C. M. Hoffmann and P. J. Vermeer. A spatial constraint problem. In J.-P. Merlet and B. Ravani, editors, *Computational Kinematics'95*, pages 83–92. Kluwer Academic Publishers, 1995.

[ILO98] ILOG. Ilog solver reference manual. Technical report, ILOG, 1998.

[JASR99] R. Joan-Arinyo and A. Soto-Riera. Combining constructive and equational constraint solving techniques. *ACM Transactions on Graphics*, 18(3):35–55, 1999.

[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.

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

[Lho93] O. Lhomme. Consistency techniques for numeric csps. In *Proc. IJCAI*, Chambery, France, 1993.

[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.