Detection and treatment of inconsistent or locally over-constrained configurations during the manipulation of 3D geometric models made of free-form surfaces

Hao Hu

N° : 2018-ENAM-0002

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

ARTS ET MÉTIERS ParisTech

École doctorale nº 432 : Sciences des Métiers de l'ingénieur

Doctorat ParisTech

THÈSE

pour obtenir le grade de docteur délivré par

l'École Nationale Supérieure d'Arts et Métiers

Spécialité doctorale "Conception"

présentée et soutenue publiquement par

Hao HU

le 23 Janvier 2018

Detection and treatment of inconsistent or locally over-constrained configurations during the manipulation of 3D geometric models made of free-form surfaces

Directeur de thèse : **Jean-Philippe PERNOT**
Co-encadrant de thèse : **Mathias KLEINER**

Jury

M. Dominique MICHELUCCI,	Professeur	Université de Bourgogne	Rapporteur
Mme Franca GIANNINI,	Senior Researcher HDR	IMATI of Genova	Rapporteur
Mme Géraldine MORIN,	Maître de conférences HDR	Université de Toulouse	Examinateur
M. Gilles CHABERT,	Maître de conférences	IMT Atlantique	Examinateur
M. Jean-Philippe PERNOT,	Professeur	Arts et Métiers ParisTech	Examinateur
M. Mathias KLEINER,	Maître de conférences	Arts et Métiers ParisTech	Examinateur

Arts et Métiers ParisTech
LABORATOIRE DES SCIENCES DE L'INFORMATION ET DES SYSTÈMES (LSIS)
UMR CNRS 7296, Centre d'Aix-en-Provencce, France

THESE

- Acknowledgments -

First of all, I would like to thank all the members of the INSM (Ingénierie numérique de systèmes mécaniques) team of the LSIS (Laboratoire de sciences de l'information et des systèmes) who explicitly or implicitly made the completion of this work a possibility.

I would like to thank all the members of the jury for the attention they have paid to my work and for their constructive remarks and questions. More precisely, I thank:

- Dominique Michelucci who made me the honor of being the president of this jury,
- Franca giannini and Dominique Michelucci who have accepted the difficult task of being my reporters,
- Gilles Chabert and Géraldine Morin who have examined carefully this manuscript.

I would like to thank my director Professor Jean-Philippe Pernot, who shared his experience in the field of free form surfaces deformation despite a very busy timetable. I have benefited a lot from the discussion with him about NURBS geometry, techniques of writing papers, work presentation etc. and I really appreciate his help during my time in LSIS. Also, I would like to thank my co-director Dr Mathias Kleiner. I have been gifted with his assistance in computer science and really appreciate him for revising my research papers.

I cannot forget all the people with whom I spend many hours and which are much more than colleagues. Merci donc à Widad, Ahmed, Romain, Yosbel, Katia, Ali, et tous les autres. Merci d'avoir pris le temps de parler lentement avec moi et d'être plus que des collègues.

And finally, I would like to thank China Scholarship Council for supporting my living expenses during my stay in France.

Hao HU

Table of contents

Résumé en Français

0.1 Introduction

De nos jours, les concepteurs s'appuient sur le logiciel de CAO 3D pour modéliser des formes complexes de forme libre basée sur les courbes et surfaces. en design industriel, cette étape de modélisation géométrique est souvent encapsulés dans un plus grand processus de développement de produit (DDP) qui peuvent comporter la conception préliminaire, l'ingénierie inverse, la simulation ainsi que les étapes de fabrication dans laquelle plusieurs acteurs interagissent (Falcidieno et al., 2014). En fait, la forme finale d'un produit est souvent le résultat d'un long et fastidieux processus d'optimisation qui vise à satisfaire les exigences associées aux différentes étapes et acteurs de la DDP. Exigences peut être vu comme contraintes. Ils sont généralement exprimés soit avec les équations, une fonction d'être réduits au minimum, et/ou en utilisant des procédures (Gouaty et al., 2016). Ce dernier se réfère à la notion de boîte noire, les contraintes n'est pas question dans le présent document, qui se concentre seulement sur les contraintes géométriques qui peuvent être exprimés par des objets linéaires ou équations non linéaires.

Pour satisfaire les exigences, les concepteurs peuvent agir sur les variables associées aux différentes étapes de la DDP. Plus précisément, dans ce document, les variables sont censés être les paramètres de la surfaces NURBS impliqués dans le processus d'optimisation de forme. Pour façonner un objet de forme libre défini par de telles surfaces, les concepteurs ont ensuite de spécifier les contraintes géométriques l'objet a à satisfaire. Par exemple, un patch doit passer par un ensemble de points 3D et de satisfaire à des contraintes de position, la distance entre deux points situés sur un patch est fixe, deux patchs doivent répondre à des contraintes de tangence ou conditions de continuité d'ordre supérieur, etc. Ces contraintes géométriques donnent lieu à un ensemble de linéaire et équations non linéaires reliant les variables dont les valeurs doivent être trouvés. En raison À l'appui local pro-

priété de NURBS (Piegl and Tiller, 1996), Les équations n'impliquent pas toutes les variables et certaines décompositions peuvent être prévus. De plus, les concepteurs peuvent exprimer involontairement plusieurs fois les mêmes exigences à l'aide de différentes contraintes, menant ainsi à des équations redondantes. Mais les concepteurs peuvent également générer des équations contradictoires involontairement et peut-être affronter avec contraintes et configurations insatisfaisant.

Parfois, des configurations avec contraintes peut être résolu par l'insertion, l'utilisation des degrés de liberté (DDL) avec le nœud de Boehm algorithme d'insertion. En conséquence, de nombreux points de contrôle sont ajoutés dans les régions où peu de ddl sont nécessaires (Pernot et al., 2005). Cette augmentation incontrôlée de la DDLs a une incidence sur la qualité générale de la finale les surfaces qui deviennent plus difficiles à manipuler que les premiers. En outre, certaines contraintes structurelles plus-ne peut pas disparaître à la suite de cette stratégie d'aide à la décision dédiés et approches doivent être développées pour identifier et gérer les configurations avec contraintes. Contrairement à 2D avancée sketchers disponible dans la plupart des logiciels de CAO, commerciale et qui peuvent identifier de manière interactive la sur-contraintes pendant le processus de dessin, il n'est pas encore tout à fait possible d'effectuer une pré-analyser l'état de la 3D à base de systèmes d'équation NURBS avant de les soumettre à un solveur. Ainsi il y a une nécessité de développer une nouvelle approche pour la détection et la résolution des contraintes redondantes et contradictoires dans les systèmes d'équation NURBS. Cela correspond à l'identification et le traitement de sur-contraint, bien limitées et sous-Pièces contraintes. Dans cet article, le traitement correspond à la suppression des contraintes avant de résoudre. Une fois les contraintes supprimées, le système d'équation devient souvent sous-contraint et le concepteur doit également ajouter une exigence par la moyenne d'une fonction d'être réduits au minimum afin de résoudre et Trouver les valeurs des inconnues. Cet aspect ne fait pas partie de l'approche proposée mais il sera discuté lors de l'introduction des résultats dans lequel un particulier est fonctionnelle réduite au minimum.

La suppression des contraintes spécifiées par l'utilisateur est une étape précédente comme le résultat ne satisfait pleinement ce que les designers ont spécifié. Ainsi, non seulement il est important d'élaborer une approche sur-mesure de supprimer les contraintes, mais il est également souhaitable de développer des mécanismes d'aide à la décision qui peuvent aider les concepteurs de cerner et d'éliminer les bonnes contraintes, c'est-à-dire ceux qui préserver autant que possible le but de la conception initiale.

Cette contribution est d'aborder ces deux questions difficiles en pro-

posant une approche d'aide à la décision d'origine pour gérer des configurations géométriques avec contraintes lorsque la déformation de surfaces de forme libre. L'algorithme linéaire poignées ainsi que d'équations non linéaires et exploite la propriété de soutien locales NURBS. S'appuyant sur une série de décompositions structurelles associées à des analyses numériques, la méthode détecte et traite aussi bien que redondante contraintes contradictoires. Depuis le résultat de ce processus de détection n'est pas unique, plusieurs critères sont mis à conduire le concepteur à identifier les contraintes qui devraient être retirés afin de limiter l'impact sur sa/son design original intention. Ainsi, même si le noyau de l'algorithme travaille sur des équations et des variables, la décision est prise en tenant compte des contraintes géométriques spécifiées par l'utilisateur à un niveau élevé.

Le papier est organisé comme suit : La section 0.2 présente le contexte et examine les travaux connexes. La section 0.3 présente le cadre de notre algorithme, énonce les principes et les caractéristiques de ses différentes étapes et propose des critères d'évaluation de ses résultats. L'approche proposée est ensuite validé sur les deux exemples académiques et industriels qui sont décrites à la section 0.4. Enfin, la section 0.5 conclut ce document par une discussion sur les principales contributions ainsi que les travaux futurs.

0.2 Contexte et travaux connexes

Cette section présente comment les concepteurs peuvent préciser leurs besoins au sein d'un problème d'optimisation. Il analyse également les méthodes utilisées pour détecter plus de structure ou numériques-contraintes.

0.2.1 Modélisation de plusieurs exigences dans un problème d'optimisation

Au cours des dernières décennies, de nombreuses techniques de déformation ont été proposés et il n'est pas le but de cet article pour détailler toutes. La plupart du temps, quand on parle de travailler sur des techniques de déformation des courbes et surfaces NURBS, l'objectif est de trouver la position X de certains points de contrôle de façon à satisfaire aux contraintes spécifiées par l'utilisateur qui peut être traduit en un ensemble de linéaire et/ou d'équations non linéaires $F(X) = 0$. car le problème est souvent à l'échelle mondiale sous-contraint, c.-à-d. il y a moins d'équations que de variables inconnues, l'un des objectifs de la fonction $G(X)$ doit également être réduit au minimum. En conséquence, la déformation des formes de forme libre est

souvent le résultat de la résolution d'un problème d'optimisation :

$$\begin{cases} F(X) = 0 \\ \min G(X) \end{cases} \tag{1}$$

Pour certaines applications particulières, le problème d'optimisation peut aussi considérer que les degrés, le nœud des séquences ou les poids des NURBS sont inconnus. Cependant, dans ce document, seule la position des points de contrôle sont considérés comme inconnus. En fonction de l'approche, l'objectif différent fonction peut être adopté, mais ils ressemblent souvent à une fonction d'énergie qui peut s'appuyer sur des modèles physiques ou mécaniques. Les contraintes boîte à outils peut également contenir des contraintes plus ou moins sophistiqué avec plus ou moins intuitive des mécanismes permettant de les définir.

Penser à la DDP ainsi qu'aux besoins de génération des formes permettant de satisfaire aux diverses exigences, l'on peut remarquer que les concepteurs ont accès à trois principaux paramètres à préciser leurs besoins et objectifs associés au sein d'un problème d'optimisation. Ils peuvent effectivement agir sur les inconnues X de décider quels points de contrôle sont fixés et quels sont ceux qui peuvent se déplacer. De cette façon, ils indiquent les parties de la forme initiale qui ne devrait pas être affecté par la déformation. Bien sûr, les concepteurs peuvent faire usage de la boîte à outils pour spécifier les contraintes les équations $F(X) = 0$ pour être convaincu. Enfin, les concepteurs peuvent également spécifier certaines de leurs exigences par la fonction $G(X)$ d'être réduit au minimum. Par exemple, ils peuvent décider de conserver ou non la forme d'origine tout en réduisant au minimum une fonction énergétique caractérisant la forme de déformation.

Cependant, la plupart des déformations de forme libre-forme techniques ne considérer que le problème résultant de l'ensemble des équations $F(X) = 0$ est sous-contraint (Elber, 2001; Bartoň, Elber, and Hanniel, 2011) et peu d'attention a été accordée à l'analyse et le traitement les sur-contraints. Cet article propose une approche pour détecter les équations redondantes et contradictoires, et d'aider le concepteur à résoudre ces problèmes par la simple élimination d'un certain nombre de contraintes. Cependant, les sections 0.3.4 et 0.4 discuter de la possibilité de fixer plus ou moins de points de contrôle et modifier ainsi le vecteur inconnu X, ainsi que la possibilité de modifier le comportement en déformation globale grâce à la personnalisation de la fonction objectif $G(X)$ d'être réduit au minimum.

0.2.2 Contraintes géométriques

Géométriques sur-contraintes sont classés structurelles et numérique sur-contraintes (Sridhar, Agrawal, and Kinzel, 1996). Structurelle sur-contraintes peut être détecté à partir d'une analyse de la DDLs, au niveau de la géométrie ou les équations. Numérique sur-contraintes sont habituellement déterminées à partir de l'analyse de la solvabilité du système d'équations. Puisque notre approche est basée sur des équations, les deux aspects sont à définir.

Contraintes structurelles

Jermann et al. donner une définition générale d'une structure avec contraintes, bien limitées et sous-systèmes d'équations limitée à un niveau macro plutôt et compte tenu de la dimension de l'espace (Jermann et al., 2006). Cette définition a été ici adapté au système d'équations où le système devrait être fixé à l'égard d'un système de coordonnées global.

Définition 1. Le degré de liberté $DDL(v)$ d'une entité géométrique v est le nombre de paramètres indépendants qui doivent être définis pour déterminer sa position et l'orientation. par exemple, dans l'espace 2D, il est égal à 2 pour les points et lignes. Pour un système de contraintes géométriques G avec un ensemble V de géométries, le degré de liberté de toutes les géométries est $DDLs = \sum_{v \in V} DDL(v)$.

Définition 2. Le degré de liberté $DDC(e)$d'une contrainte géométrique e est le nombre d'équations indépendantes nécessaires pour représenter. Par exemple, les contraintes de distance ont une DDC en 2D et 3D. Pour un système de contraintes géométriques G avec un ensemble E of contraintes, le degré de liberté de l'ensemble des contraintes est $DDCs = \sum_{e \in E} DDC(e)$.

Définition 3. Un système de contraintes géométriques G est *structurelle bien-contraint* si G satisfait $DDCs = DDLs$ et si tous les sous-systèmes après décomposition satisfaire $DDCs \leq DDLs$.

Définition 4. Un système de contraintes géométriques G est *structurelle sur-contraint* s'il existe un sous-système satisfaisant $DDCs > DDLs$. *Structurelle sur-contraint* sont les contraintes que transformer un *structurelle sur-contraint* système dans un *structurelle bien-contraint* system lorsqu'ils sont supprimés.

Enfin, les définitions de comptage basées sur le DDL-comparer le nombre d'équations pour le nombre de variables d'un système. Cependant, il n'est pas couvrir les cas tels que les licenciements géométriques induites par les théorèmes géométriques. Afin de couvrir ces situations, les définitions

algébriques sont introduits.

Sur-contraintes numériques

Les définitions structurelles antérieures ne peuvent pas distinguer les contraintes redondantes et contradictoires. Cependant, du point de vue algébrique, c'est que cet examen pourrait être traitée correctement par Grobner ou méthodes Wu-Ritt (Chou and Gao, 1990). Ces méthodes sont couramment utilisées dans l'algèbre abstraite et requiert de solides bases mathématiques pour comprendre.

Définition 5. Soit $G = (E, V, P)$ un système de contraintes géométriques, où E est un ensemble d'équations, V est un ensemble de variables et P est un ensemble de paramètres. E_r est une collection non-vide de sous-ensembles de E, appelée textbf equations de base (nous l'appelons **base** en bref, satisfaisant:

- non **base** contient correctement un autre **base**;
- si E_{r1} et E_{r2} sont **base** respectivement et si e est une équation de E_{r1}, alors il y a une équation f de E_{r2} tel que $\{(E_{r1} - e) \cup f\}$ est aussi un **base**.

Définition 6. Soit $G = (E, V, P)$ un système de contraintes géométriques. Soit E_r un **base**. Pour une équation e, en l'ajoutant à E_r formant un nouveau groupe: $\{E_r \cup e\}$. Si $\{E_r \cup e\}$ est solvable, alors e est une **équation redondante**.

Définition 7. Soit $G = (E, V, P)$ un système de contraintes géométriques. Soit E_r un **base**. Pour une équation e, en l'ajoutant à E_r formant un nouveau groupe: $\{E_r \cup e\}$. Si $\{E_r \cup e\}$ n'est pas solvable, alors e est une **équation conflictuelle**.

Définition 8. Soit $G = (E, V, P)$ un système de contraintes géométriques composé de deux sous-systèmes: $G_b = (E_b, V, P)$ et $G_o = (E_o, V, P)$ avec $\{E = E_b \cup E_o, E_b \cap E_o = \varnothing\}$. Si E_b est un **base**, alors E_o est un ensemble de **sur-contraintes numériques**.

Il convient de noter que l'ensemble des *contraintes de base* and *numérique sur-contraintes* d'un système donné n'est pas unique. Ainsi, décision-support des mécanismes et des critères doivent être définis pour aider les concepteurs à identifier le bon redondant et à supprimer les contraintes contradictoires.

0.2.3 Modélisation du système de contraintes géométriques

Tel que discuté précédemment, un système de contraintes géométriques peuvent être décrites, que ce soit au niveau des équations ou au niveau de la géométrie. D'une part, d'un système d'équations, il existe des méthodes algébriques en mesure de s'attaquer directement aux problèmes de cohérence (Cox, Little, and O'Shea, 2015) ou d'analyser la structure indirectement à partir d'un graphe bipartite où deux classes de nœuds représentent des variables et équations indépendamment (Bunus and Fritzson, 2002a). D'autre part, pour la modélisation au niveau de la géométrie, deux types de graphique sont principalement utilisés : soit les graphes bipartis avec deux classes de nœuds représentant les entités géométriques et contraintes séparément (Hoffmann, Lomonosov, and Sitharam, 1998), ou les graphiques de contrainte avec les noeuds représentant les entités géométriques et les bords des contraintes représentant (Gao and Chou, 1998; Hoffman, Lomonosov, and Sitharam, 2001).

Toutefois, compte tenu des systèmes NURBS de contrainte n'est pas simple car il existe plusieurs types de variables contribuant à la forme de déformation. d'une part, système d'équations d'activer un moyen viable de paramètres de modélisation où comme noeuds et poids peuvent être définies comme des variables. D'autre part, la modélisation graphique au niveau de la géométrie est actuellement limitée à des variables comme les coordonnées des points de contrôle et les poids qui leur sont associés. Représentant des variables telles que degrés, noeuds, les valeurs de u et v paramètres à l'aide de graphes de contraintes au niveau de la géométrie n'est pas démontré de façon convaincante dans la littérature. Dans l'œuvre de Lesage (Lesage, 2002), les points de contrôle entre les vecteurs sont utilisés pour représenter des objets Nurbs ainsi que les contraintes géométriques telles que l'incidence ou de tangence. Néanmoins, sa méthode est limitée aux cas où les points de contrôle sont inconnues seulement et n'est pas suffisamment général pour la comparaison de la modélisation à l'équation.

Selon les définitions précédentes, les méthodes de détection peuvent être classées en deux catégories (Sridhar, Agrawal, and Kinzel, 1996): structurelle sur-contraintes et numérique sur-contraintes détection.

0.2.4 Structurelle sur-contraintes détection

De nombreuses structurelle sur-contraintes peuvent être identifiés en comptant DDLs au cours de la processus de décomposition du système. Parmi les méthodes de décomposition, ceux qui sont utiles pour trouver avec contraintes structurellement sous-parties ont été identifiés et classés en trois

catégories selon le type de sous-systèmes.

Modèles spécifiques

Cette catégorie est basée sur le fait que, lors de l'examen de leurs plans d'ingénierie, la plupart des systèmes peuvent être décomposés tout en reconnaissant une configuration spécifique, construit par règle et compas. division récursive est d'abord proposé par Owen pour gérer les systèmes de contraintes 2D où seule la distance et l'angle les contraintes sont impliqués (Owen, 1991). Il a introduit plusieurs règles pour la détection de la redondance, y compris des règles pour détecter les sous-parties trop rigide et règles pour vérifier si les licenciements d'angle (Owen, 1996). Fudos et Hoffman a adopté la méthode de réduction graphique qui fonctionne bien avec les systèmes bien-contraint et sur-contraint en 2 dimensions (Fudos and Hoffmann, 1997). Ces méthodes sont en temps polynomial mais pas assez général en raison du peu de répertoire de modèles, qui ne peut pas couvrir tous les types de configurations géométriques.

Rigidité structurelle

Cette classe regroupe les méthodes qui se décomposer un système en sous-systèmes rigides structurelles. Les méthodes varient en fonction de la structure adoptée-rigidité définition ainsi que sur les algorithmes de recherche correspondant. En modifiant le débit maximal du réseau supplémentaires théorie, Hoffmann et al. développé la *Dense* algorithme pour identifier les 1-bien-contraint sous-graphe (Hoffmann, Sitharam, and Yuan, 2004). Leur définition des sous-système rigide découle de Laman's théorème sur la caractérisation de la rigidité des cadres de bar (*Combinatorial Rigidity*). Toutefois, la définition n'est pas traiter correctement les contraintes telles que les incidences et parallélismes, qui sont largement utilisés dans les systèmes de CAO. Jermann et al. modifié leur définition en introduisant la notion de *degree of rigidity* (DoR) pour remplacer la dimension D (Jermann, Neveu, and Trombettoni, 2003). La différence entre les deux réside sur le fait que la valeur de DoR varie en sous-systèmes alors que d reste constante quelle que soit la sous-systèmes sont (par exemple en 2D, D =3 et en 3D, D =6). Basée sur la même théorie du flux réseau, son algorithme *Over-rigid* traite correctement les contraintes et/ou l'incidence parallèle spécifié mais est encore limitée à des conditions où l'incidence les dégénérescences générique en raison de théorèmes géométriques comme le co-linéarités, co-planarities sont interdits (Jermann, Neveu, and Trombettoni, 2003).

Correspondance maximum

Ces méthodes reconnaître les modèles avec contraintes structurelles en comparant directement les DDLs et DDCs d'un système (ou sous-système) sans prendre en compte la dimension constante dépendante D. Dulmage-Mendelsohn (D-M) de l'algorithme de décomposition permet de décomposer un système d'équations en sur-contraintes, bien-contraintes, et sous-contraintes sous-systèmes (Dulmage and Mendelsohn, 1958). Il a été utilisé pour le débogage dans l'équation de la modélisation des systèmes tels que Modelica (Bunus and Fritzson, 2002b). Par ailleurs, il calcule un graphique acyclique (DAG) qui offre une résolution de l'ordre parmi les composantes fortement connectées (SCC) du système. Serrano a été intéressé par l'utilisation de l'algorithme de la théorie des graphes avec contraintes pour empêcher les systèmes où toutes les contraintes et les entités géométriques sont d'un DDL (Serrano, 1987). Latham et al. a étendu le travail de Serrano en proposant la comparaison pondérée b maximum pour identifier les contraintes à l'arbitraire avec DDLs (Latham and Middleditch, 1996).

Une partie avec contraintes contient soit redondant ou contraintes contradictoires. Mais les autres parties peuvent également contenir des sur-contraintes de numérique (Podgorelec, Žalik, and Domiter, 2008). C'est parce que l'analyse ne tient pas compte de la structure de l'information numérique d'un système. Par conséquent, pour mieux identifier les contraintes comme les plus subtils de la redondance géométrique, méthodes numériques doivent être adoptées.

0.2.5 Détection numérique de sur-contraintes

Toute contrainte géométrique peut être transformé en un ensemble d'équ ations algébriques (Hoffmann, Lomonosov, and Sitharam, 1998). Par conséqu ent, géométriques sur-contraints sont l'équivalent d'un ensemble de contradiction or d'équations redondantes. Ici, les méthodes de détection numérique ont été classées en deux catégories selon le type de contraintes.

Détection de sur-contraintes linéaires

L'élimination de Gauss, factorisation LU avec pivot partiel et Factorisation QR avec pivotant colonne ont été adoptées avec succès pour trouver des équations contradictoires/redondants ainsi que de groupe dans les systèmes d'équations linéaires (Strang, 2006). Light and Gossard a appliqué l'élimination de Gauss pour calculer le rang ainsi que pour identifier davantage les équations invalides (Light and Gossard, 1983). Serrano a étendu

son travail pour vérifier l'existence de sur-contraintes dans les composants fortement liés d'un système d'équations (Serrano, 1991). Ces méthodes ont permis des détections stables et rapides mais se limitent aux cas linéaires.

Détection non-linéaire des sur-contraintes

Méthodes symboliques sont théoriquement fiables mais la complexité du temps est exponentielle. Kondo a utilisé la méthode de base de Grobner pour tester la dépendance entre les contraintes de dimension 2D (Kondo, 1992). Gao et Chou ont présenté l'algorithme de décomposition de Wu-Ritt pour déterminer si un système est trop contraint (Gao and Chou, 1998). Cependant, les deux méthodes ne trouvent pas directement les groupes d'extension de contraintes excessives.

Méthodes d'optimisation ont été utilisées pour résoudre les problèmes de satisfaction des contraintes, qui fonctionne bien pour les systèmes sous contraintes (Ge, Chou, and Gao, 1999).

Méthodes d'analyse matricielle jacobienne permettent une détection plus rapide en étudiant la structure jacobienne des équations. Cependant, ils ne sont pas en mesure de distinguer les contraintes redondantes et conflictuelles. La principale différence entre ces méthodes est la configuration où la matrice jacobienne devrait se développer. Si le système est résoluble, Haug a proposé de perturber la racine commune et de recalculer le rang une fois que la matrice jacobienne est déficiente (Haug, 1989). Cependant, si le système n'est pas résoluble, Foufou et al. suggèrent une méthode probabiliste numérique (NPM), qui analyse la matrice jacobienne à des configurations aléatoires (Foufou and Michelucci, 2012). Cependant, il existe un risque que la matrice jacobienne soit classée en défaut aux points choisis, mais elle est à plus grande échelle partout ailleurs. Par conséquent, NPM est pratique dans le calcul mais peut conduire à des contraintes excessives détectées incorrectement. Au lieu de sélectionner au hasard les configurations, Michelucci et al. Ont suggéré d'étudier la structure jacobienne à la configuration des témoins où les contraintes d'incidence sont satisfaites (Michelucci et al., 2006). La witness configuration et la cible configuration partage la même structure jacobienne. En conséquence, tous les sur-contraints sont identifiées. Plus récemment, Moinet et al ont développé des outils pour identifier des contraintes conflictuelles en analysant le témoin d'un système linéarisé de les équations (Moinet, Mandil, and Serre, 2014). Leur approche a été appliquée au cas de test de double banane bien connu sur lequel notre approche sera également testée dans la section 0.4.

À partir de la discussion ci-dessus, il est clair que différentes méthodes

sont capables de gérer certains systèmes de contraintes géométriques. Cependant, aucune méthode ne peut couvrir tous les cas parfaitement selon nos critères. Ainsi, dans cet article, une nouvelle approche qui couple structurelle comme ainsi que des méthodes numériques sont proposées.

0.3 Une approche générique couplant les décompositions structurelles et les analyses numériques

Cette section décrit notre approche pour détecter et traiter les contraintes géométriques redondantes et conflictuelles. L'idée principale est de décomposer le système d'équations en petits blocs qui peuvent être analysés itérativement en utilisant des méthodes numériques dédiées. Le cadre global et l'algorithme sont introduits avant de préciser les différentes étapes impliquées.

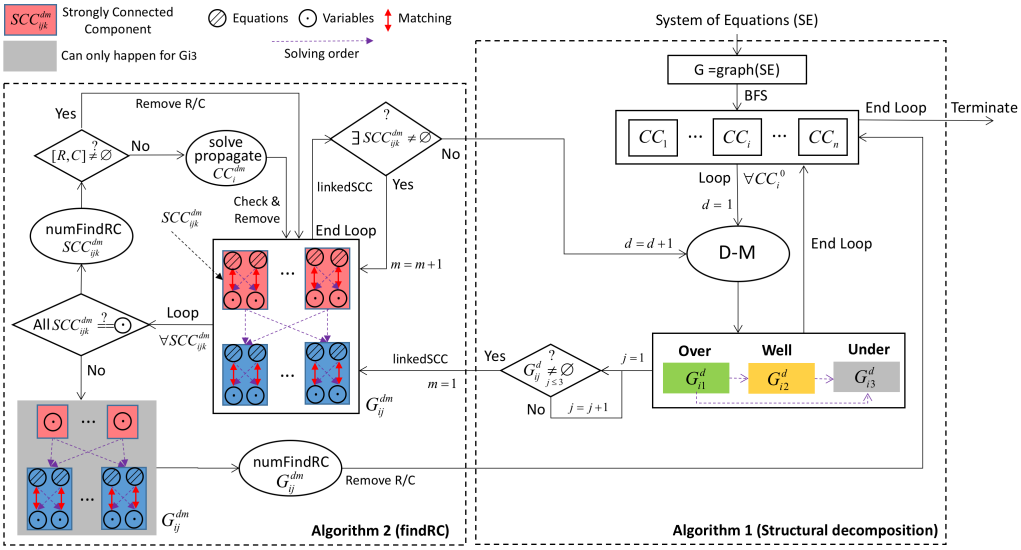


Figure 1: Cadre global composé de trois boucles imbriquées définissant la structure principale de l'algorithme de détection.

0.3.1 Cadre de détection global

Le cadre général a été modélisé en figure 1. Il est basé sur trois boucles imbriquées: la décomposition structurelle en composants connectés (CC); la décomposition structurelle d'un CC dans ses sous-parties ($G1, G2, G3$) et son DAG correspondant de composants fortement connectés (SCC); l'analyse numérique itérative de ces SCC. Le pseudo-code pour les procédures principales est fourni dans la section 0.3.2.

Boucle parmi les composants connectés

Le système des équations (SE) est initialement représenté par une structure de graphique G, où les noeuds correspondent aux variables et aux arêtes aux équations. La structure est d'abord décomposée en n components connectés $\{CC_1, \cdots, CC_n\}$ à l'aide de Breadth First Search (BFS) (Leiserson and Schardl, 2010). Une telle décomposition est rendue possible grâce à la propriété de support local de NURBS ou tout simplement en utilisant des contraintes qui dissocient quoi se produit selon les instructions x, y et z du cadre de référence (p. ex. contraintes de position ou de coïncidence). Par conséquent, les contraintes géométriques peuvent être détectées séparément pour chaque CC_i.

Boucle parmi les sous-parties obtenues par décomposition D-M

La décomposition de DM est utilisée pour décomposer structurellement CC_i en un maximum de trois sous-parties: G_{i1} (sous-partie sur-contrainte), G_{i2} (sous-partie bien contrainte) et G_{i3} (sous-sous-contrainte). Chaque sous-partie (si elle existe) sera analysée itérativement à l'aide de la troisième boucle imbriquée expliquée ci-dessous.

Cependant, un seul passage de la troisième boucle sur chaque G_{ij} n'est pas suffisant. En effet, toute passe peut conduire à la suppression des contraintes, qui modifie la structure de CC_i et nécessite donc d'appliquer la décomposition D-M à nouveau après la passe pour obtenir des sous-parties mises à jour. L'exposant d est utilisé pour noter que CC_i^d (resp. G_{ij}^d) se réfère à CC_i (resp. G_{ij}) après son d^{th} DM. Bien que le nombre de passes requises soit inconnu à l'avance, il est garanti que le processus converge vers un état où seule une sous-partie G_{i3} est restée. En d'autres termes, les contraintes seront supprimées ou déplacées vers la troisième sous-partie le long du processus.

Boucle parmi les composants fortement connectés

En plus des sous-parties, la décomposition de DM fournit également un DAG pour chaque CC_i^d. Les nœuds de ce DAG sont des composants fortement connectés SCC_{ijk}^d. Bordes de ce DAG (violet les flèches dans la figure 1) désignent la résolution des dépendances entre SCC_{ijk}^d et peuvent traverser les sous-paragraphes G_{ij}^d limites. Dans ce qui suit, linkedSCC (G_{ij}^d) se réfère à l'opération qui obtient (la sous-partie de) ce DAG à partir de d^{th} DM décomposition de CC_{ij}^d qui correspond à la sous-partie donnée.

La troisième boucle consiste à essayer itérativement (dans l'ordre induit par les dépendances DAG) de trouver des contraintes numériques dans chaque SCC^d_{ijk} ou, lorsqu'il est résoluble, propager sa solution à d'autres blocs. Étant donné que les blocs sont fortement connectés, il n'y a qu'une seule solution possible pour chaque bloc, à moins qu'il contienne uniquement des variables, et ce dernier cas ne peut être rencontré que dans une troisième sous-partie G^d_{i3}. Le processus ne fonctionne que le niveau supérieur du DAG (blocs rouges de la figure) car ces équations de blocs n'utilisent pas de variables d'autres blocs.

Pour chaque bloc rouge, et comme indiqué dans la partie supérieure gauche de figure refdh, une méthode numérique appropriée (numFindRC dans la figure et le pseudo-code) essaie de trouver redondant (R) ou conflictuelles (C). Ces sur-contraintes sont ensuite supprimées du composant connecté actuellement analysé CC^d_{ij}. Si le bloc est résoluble, sa solution (unique) se propage vers des blocs dépendants, ce qui peut entraîner la détection de contraintes supplémentaires redondantes ou conflictuelles, et supprimées de CC^d_{ij}. Une fois que tous les blocs rouges ont été analysés, cette partie du DAG (potentiellement transformer les blocs bleus en rouge) est recalculée jusqu'à ce que tous les blocs soient analysés. Cependant, il n'est pas nécessaire de recalculer la décomposition D-M sur l'ensemble de CC^d_{ij}; il suffit de la recalculer uniquement pour l'équivalent maximal pour la sous-partie actuelle en appelant à nouveau le service linkedSCC. L'exposant m est utilisé pour noter que G^{dm}_{ij} (resp. SCC^{dm}_{ijk}) se réfère à G^d_{ij} (resp. SCC^d_{ijk}) après sa m^{th} matching. Bien que le nombre de passes requises soit inconnu à l'avance, il est garanti que le processus converge vers un état où il n'y a plus de blocs, ou ces blocs ne contiennent que des variables (et cela n'est possible que pour la troisième sous-partie G^d_{i3}). En d'autres termes, les contraintes et les variables sont supprimées jusqu'à ce que nous obtenions un système sous-contraint avec plusieurs solutions, ce qui signifie qu'il n'y a plus de propagation possible. Dans cette dernière étape, comme indiqué dans la partie inférieure gauche de la figure, le système restant est analysé pour les conflits numériques et procède avec le prochain composant connecté.

0.3.2 Pseudo-code

Cette section fournit le pseudo-code pour les deux procédures principales de l'approche, entouré de rectangles pointillés sur la figure 1.

Algorithm 1 Structural decomposition

SE ← System of Equations
$G \leftarrow$ Graph(SE)
$[CC_1, \cdots, CC_n] \leftarrow$BFS($G$)
for $i = 1$ to n **do**
  $[G^1_{i1}, G^1_{i2}, G^1_{i3}] \leftarrow$DM($CC_i$)
  $CC^1_i \leftarrow CC_i$
  for $j = 1$ to 3 **do**
    $d \leftarrow 1$
    continue$\leftarrow$True
    while continue & $G^d_{ij} \neq \varnothing$ **do**
      continue,$CC^{d+1}_i \leftarrow$ findRC(CC^d_i, G^d_{ij})
      $d \leftarrow d + 1$
      $[G^d_{i1}, G^d_{i2}, G^d_{i3}] \leftarrow$DM($CC^d_i$)
    end while
  end for
end for
return $[CC^d_1, \cdots, CC^d_n]$

Algorithm 2 findRC: Numerical analysis of G^d_{ij} subpart of CC^d_i

Require: CC^d_i and G^d_{ij}
Ensure: Boolean continue and updated CC^d_i
$[SCC^{d1}_{ij1}, \cdots, SCC^{d1}_{ijN}] \leftarrow$linkedSCC($G^d_{ij}$)
$m \leftarrow 1$
$G^{d1}_{ij} \leftarrow G^d_{ij}$
while $[SCC^{dm}_{ij1}, \cdots, SCC^{dm}_{ijN}] \neq \varnothing$ **do**
  $l \leftarrow 0$
  for $k = 1$ to N **do**
    if onlyVariable(SCC^{dm}_{ijk}) **then**
      $l \leftarrow l + 1$
    else
      $[R, C] \leftarrow$ numFindRC(SCC^{dm}_{ijk})
      if $[R, C] == \varnothing$ **then**
        solution $\leftarrow$ solve(SCC^{dm}_{ijk})
        propagate(solution,CC^d_i)
        $R \leftarrow$ checkRedundant(CC^d_i)
        $C \leftarrow$ checkConflicting(CC^d_i)
      end if
      $CC^d_i \leftarrow$ removeRCfromCC($CC^d_i, [R, C]$)
    end if
    if $l == N$ **then** ▷ all red blocks contain only variables
      $[R, C] \leftarrow$ numFindRC(CC^d_i)
      $CC^d_i \leftarrow$ removeRCfromCC($CC^d_i, [R, C]$)
      return False,CC^d_i
    end if
  end for
  $G^{d(m+1)}_{ij} \leftarrow$ update(CC^d_i)
  $m \leftarrow m + 1$
  $[SCC^{dm}_{ij1}, \cdots, SCC^{dm}_{ijN}] \leftarrow$linkedSCC($G^{dm}_{ij}$)
end while
return True,CC^d_i

0.3.3 Analyse des composants fortement connectés

Cette section traite des techniques utilisées pour analyser les composants fortement connectés SCC_{ijk}^{dm}. Ceci correspond à la fonction numFindRC de Algorithme 2 (section 0.3.2) utilisé pour trouver les contraintes redondantes (R) et conflictuelles (C) d'un composant s'il existe. Sinon, le composant est résolu et les solutions sont propagées sur l'ensemble du système.

Selon le type de contraintes, c'est-à-dire linéaire ou non linéaire, les méthodes diffèrent et sont présentées dans les sous-sections suivantes. La notation suivante $A[i:j, l:k]$ est utilisée pour définir la matrice obtenue en coupant les i th à j th lignes et les l th pour kth colonnes de A.

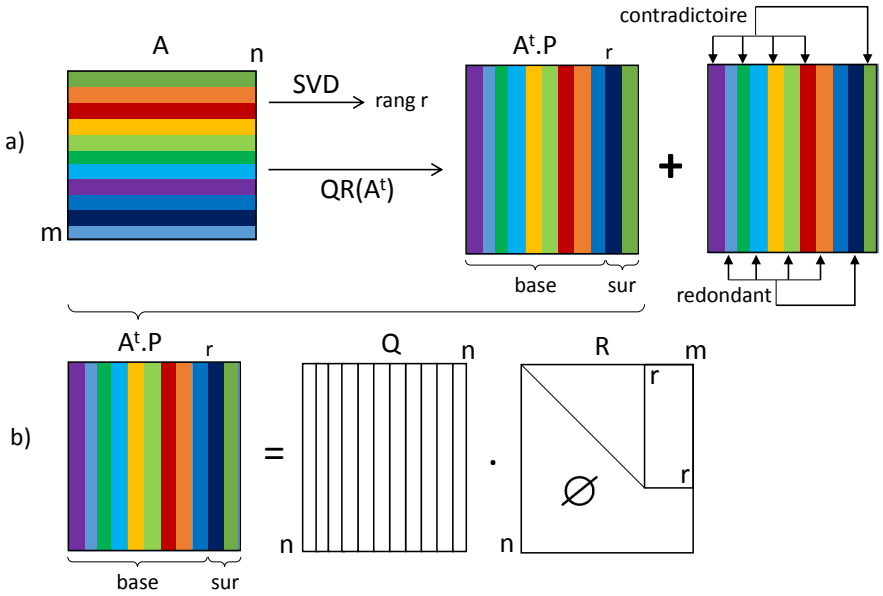


Figure 2: Analyse par blocs des systèmes linéaires: (a) processus global de détection, (b) factualisation QR avec pivotement des colonnes.

Système linéaire

Dans l'approche proposée, la factorisation QR avec le pivotement des colonnes est utilisée pour détecter les contraintes linéaires. La factorisation QR avec une permutation de colonne facultative P, déclenchée par la présence d'un troisième argument de sortie, est utile pour détecter la singularité ou la carence de rang la figure 2 montre le processus de détection global. Les lignes droites de couleur horizontale correspondent aux équations linéaires du système $Ax = b$ à résoudre, où A a une dimension $m \times n$. Ici, on suppose que le rang du système est r, ce qui signifie qu'il y a des équations indépendantes de r avec $m > r$. Le rang est calculé à l'aide de SVD, ce qui

est relativement stable par rapport à d'autres méthodes.

En ce qui concerne la factorisation QR, les colonnes sont échangées au début de k ième étape pour s'assurer que:

$$\left\|A_k^{(k)}(k:m))\right\|_2 = \max_{j \geqslant k} \left\|A_j^{(k)}(k:m))\right\|_2 \tag{2}$$

où $A_j^{(k)}(k:m) = A[k:m, j]$. À chaque étape de la factorisation, la colonne de la matrice non factorisée restante avec la plus grande norme est utilisée comme base pour cette étape et est déplacé vers la position principale (Golub and Van Loan, 2013). Cela garantit que les éléments diagonaux de R se produisent en ordre décroissant et que toute dépendance linéaire parmi les colonnes est certainement révélée en examinant ces éléments. La matrice de permutation P réarrange les colonnes de A^t afin que les colonnes apparaissent dans l'ordre décroissant de leur norme.

Les premières r colonnes de A^tP sont les contraintes de base de A^t et les premières r colonnes de Q form une base orthogonale (figure 2.b). Puisque les colonnes mr restantes dépendent linéairement des premières r columns (Dongarra and Supercomputing, 1990), elles sont les contraintes excessives. Le rang r correspond également au nombre de valeurs non nulles d'éléments diagonaux de R.

Pour trouver des dépendances linéaires entre les colonnes, la déduction suivante est nécessaire. D'abord, la matrice $Q(:, 1:r)$ est inversée en utilisant l'équation suivante:

$$A^t(:, 1:r) = Q(:, 1:r).R(1:r, 1:r)$$

et est ensuite utilisé dans l'équation suivante:

$$A^t(:, r+1:n) = Q(:, 1:r).R(1:r, r+1:n)$$

fournissant ainsi la relation suivante entre les deux matrices tranchées $A^t(:, r+1:n)$ and $A^t(:, 1:r)$:

$$A^t(:, r+1:n) = A^t(:, 1:r).R(1:r, 1:r)^{-1}R(1:r, r+1:n)$$

Enfin, pour identifier les équations redondantes et contradictoires, le nouveau b vector après factorisation est redéfini comme suit:

$$b_{new} = b(r+1:n) - b(1:r).R(1:r, 1:r)^{-1}R(1:r, r+1:n)$$

Les équations redondantes et contradictoires sont encore distinguées en comparant la valeur des derniers $m - r$ éléments de b_{new} avec 0.

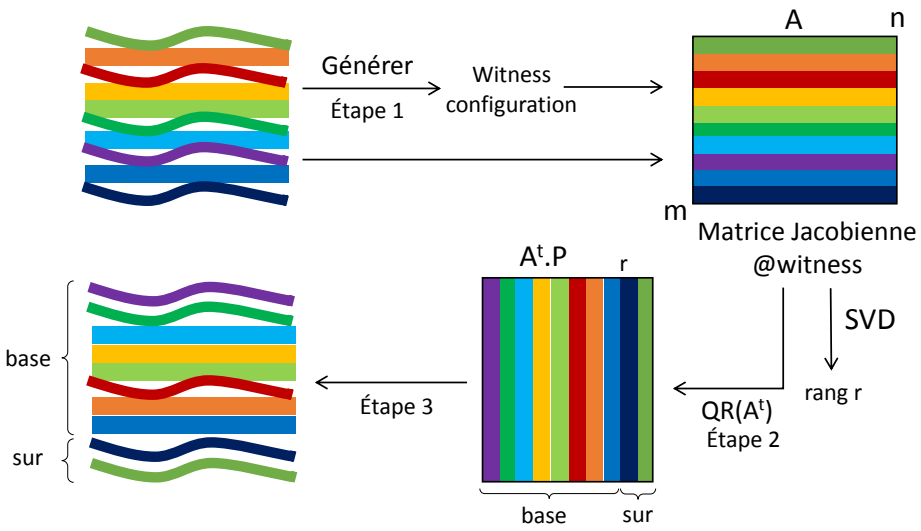


Figure 3: Analyse séquentielle des systèmes non linéaires. Phase I: détection des sur-contraintes .

Système non-linéaire

En considérant un système d'équations non linéaires, on utilise un processus d'identification en deux phases. Tout d'abord, la Witness Configuration Method (Michelucci et al., 2006) est utilisé pour trouver toutes les contraintes excessives (phase I), et Grobner Basis ou Incremental Solving est ensuite appliqué pour distinguer davantage les contraintes redondantes et contradictoires (phase II).

Phase I. Prenant avantage de la méthode proposée par Moinet et al. (Moinet, Mandil, and Serre, 2014), une configuration de témoin générique est générée à partir de la forme initiale de l'objet à déformer (étape 1). Effectivement, dans notre cas, les variables x sont les positions des points de contrôle qui ont un emplacement initial $x^{(0)}$ avant déformation. Ensuite, la factorisation QR avec le pivotement de colonne est utilisée pour analyser cette configuration de témoin (étape 2)., la séquence des équations est réorganisée. Le premier r (le rang des équations Jacobian matrix est indépendant tandis que les autres sont les sur-contraintes. Dans la figure 3, les lignes courbes courbes représentent des équations non linéaires.

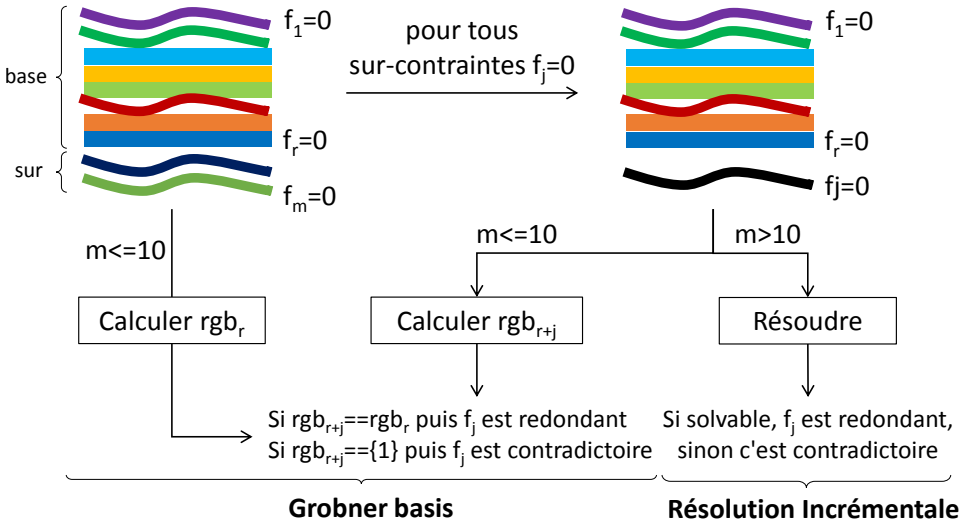


Figure 4: Phase II: Distinguer des contraintes redondantes et conflictuelles

Phase II. Pour mieux distinguer les contraintes redondantes et conflictuelles, on utilise soit la Grobner Basis, soit la résolution incrémentale. Dans notre algorithme, ce choix repose sur le nombre d'équations. Si le nombre d'équations $m \leq 10$, on préfère Grobner Basis (Lamure and Michelucci, 1998). Sinon, la résolution incrémentale est choisie. Pour expliquer les deux méthodes, supposons que le système de contraintes suivant est disponible après la phase I (figure 4):

$$\begin{cases} f_1(x_1, x_2, \cdots, x_n) = 0 \\ f_2(x_1, x_2, \cdots, x_n) = 0 \\ \vdots \\ f_r(x_1, x_2, \cdots, x_n) = 0 \\ f_{r+1}(x_1, x_2, \cdots, x_n) = 0 \\ \vdots \\ f_m(x_1, x_2, \cdots, x_n) = 0 \end{cases} \tag{3}$$

où les équations 1 à r sont les *contraintes de base* et les équations $(r+1)$ à m sont les *sur-contraintes*.

La méthode de résolution incrémentielle insère de façon incrémentielle la sur-contrainte $f_j = 0$, $j \in \{r+1, \cdots, m\}$, dans l'ensemble des contraintes de base formant ainsi un nouveau groupe d'équations $\{f_1 = 0, \cdots, f_r = 0, f_j = 0\}$. Si le nouveau groupe est résoluble, alors l'équation $f_j = 0$ est redondante, sinon elle est contradictoire. Bien sûr, les contraintes de base sont toujours résolvables.

Quand Grobner basis (Cox, Little, and O'Shea, 2015) sont utilisés, la méthode calcule d'abord la réduction Grobner basis rgb_r de l'idéal $\langle f_1, \cdots, f_r \rangle$. Puisque l'ensemble des équations sont résolubles, $rgb_r \neq \{1\}$. Ensuite, une boucle sur toutes les sur-contraintes $f_j = 0$, $j \in \{r+1, \cdots, m\}$, réduction Grobner basis rgb_{r+j} de l'idéal $\langle f_1, \cdots, f_r, f_j \rangle$ est calculé. Si $rgb_{r+j} \equiv rgb_r$, alors $f_j = 0$ est une équation redondante. Si $rgb_r \subset rgb_{r+j}$, alors $f_j = 0$ est une équation redondante. Finalement, si $rgb_{r+j} = \{1\}$, alors $f_j = 0$ est une équation conflictuelle.

0.3.4 Validation et évaluation des solutions

La section 0.2.1 a introduit les multiples façons de modéliser les exigences dans un problème d'optimisation en spécifiant un vecteur inconnu X, les contraintes à satisfaire $F(X) = 0$ et la fonction $G(X)$ pour minimiser.

L'approche décrite dans cette section permet d'identifier des équations redondantes et contradictoires. L'exactitude est assurée puisqu'elle consiste en un algorithme à virgule fixe qui ne s'arrête que lorsque le système est résoluble. De plus, toute équation supprimée est garantie soit conflictuelle ou redondant avec l'ensemble restant. On a donc montré que l'ensemble des équations $F(X) = 0$ peut être décomposé en deux sous-ensembles: $F_b(X) = 0$ contenant les équations de base, $F_o(X) = 0$ les sur-contraintes.

Pour rester proche des exigences que le concepteur a en tête, l'approche proposée passe alors du niveau des équations au niveau des contraintes. Ainsi, les contraintes géométriques associées aux équations $F_o(X) = 0$ sont analysées et toutes les équations liées à ces contraintes sont regroupées dans un nouvel ensemble d'équations $\widetilde{F}_o(X) = 0$. Ainsi, les contraintes géométriques associées aux équations $F_o(X) = 0$ sont analysées et toutes les équations liées à ces contraintes sont rassemblées dans un nouvel ensemble d'équations $\widetilde{F}_o(X) = 0$. Enfin, les équations liées aux contraintes qui ne sont ni conflictuelles ni redondantes forment l'autre ensemble $\widetilde{F}_b(X) = 0$. Cette transformation permet de travailler au niveau des contraintes et non au niveau des équations. Ceci est beaucoup plus pratique pour l'utilisateur final intéressé à travailler au niveau des exigences géométriques.

Puisque cette décomposition n'est pas unique, elle donne naissance à diverses solutions finales potentielles (la décomposition interactive est hors de portée de cet article). Plusieurs critères sont maintenant introduits pour évaluer ces solutions en fonction de l'intention initiale de conception. Caractériser la qualité des solutions obtenues, l'ensemble des paramètres spécifiés par l'utilisateur P est introduit Cet ensemble rassemble tous les paramètres que le concepteur peut introduire pour définir les contraintes que sa forme

doit satisfaire. Par exemple, la distance d imposée entre deux points d'une surface NURBS est un paramètre caractérisant une partie de l'intention de conception. Ensuite, l'idée est d'évaluer dans quelle mesure les solutions s'écartent de l'intention initiale de conception et notamment en termes de paramètres P.

Pour ce faire, le problème d'optimisation contenant les contraintes de base est résolu:

$$\begin{cases} \widetilde{F}_b(X) = 0 \\ \min G(X) \end{cases} \tag{4}$$

et la solution X' est alors utilisée pour évaluer les sur-contraintes non satisfaites $\widetilde{F}_o(X')$ ainsi que les valeurs réelles P' des paramètres P spécifiés par l'utilisateur. Par exemple, si la distance spécifiée par l'utilisateur d entre les deux patches ne peut pas être atteinte, la distance réelle d' sera mesurée sur la solution obtenue. A partir de cette solution, il est possible d'évaluer trois Critères:

- *Déviation en termes de paramètres/contraintes*: ce critère vise à mesurer dans quelle mesure les valeurs réelles P' des paramètres proviennent des paramètres spécifiés par l'utilisateur P. Ce critère aide à comprendre si l'intention de conception est préservée en termes de paramètres et par conséquent en termes de contraintes.

$$df = \frac{\sum_i |P'_i - P_i|}{\sum_i |P_i|} \tag{5}$$

- *Déviation en termes de fonction pour minimiser*: ce critère évalue directement dans quelle mesure la fonction objective G a été minimisée. Ici, la fonction est simplement calculée à partir de la solution X' du problème d'optimisation. Pour préserver l'intention de conception, cette valeur doit être minimisée. Ainsi, il peut être utilisé pour comparer les solutions entre eux.

$$dg = G(X') \tag{6}$$

- *Degré de quasi-dépendance*: la carence de rang de la matrice jacobienne sur le témoin révèle clairement les dépendances entre contraintes. Cependant, pour les systèmes d'équations basés sur NURBS, les contraintes peuvent être indépendantes mais proches d'être dépend antes. Dans ce cas, la matrice jacobienne de $\widetilde{F}_b(X)$ au point de solution X' est mal conditionnée et la solution correspondante peut être de mauvaise qualité Le troisième critère évalue donc le nombre de

condition (*cond*) de la matrice jacobienne comme mesure de la quasi-dépendance (Kincaid and Cheney, 2002):

$$cond = cond(\mathbb{J}_{\widetilde{F}_b}(X')) \tag{7}$$

Enfin, même si ces critères caractérisent la qualité de la solution X' par rapport à l'intention de conception, ils n'ont pas été combinés dans un indicateur unique. Ainsi, les résultats de la section suivante seront évalués en analysant et en comparant trois critères pour chaque solution.

0.4 Résultats et discussion

Cette section présente deux configurations sur lesquelles la technique de détection et de résolution de sur-contraintes proposée a été testée: Le premier concerne le cas académique de double banane largement étudié dans la littérature Il a été utilisé pour comparer notre solution à celles générées par d'autres. Le second exemple est plus industriel et concerne la mise en forme d'un verre composé de plusieurs patchs NURBS.

0.4.1 Cas de test Double-Banane

Les variables X, les contraintes $F(X) = 0$ et les paramètres P du cas de test Double Banane sont exactement les mêmes que ceux testés par Moinet et al. (Moinet, Mandil, and Serre, 2014). La seule différence est qu'ils utilisent une formulation sans coordonnées tandis que la nôtre est à base cartésienne. Ici, l'objectif est de trouver la position des 8 nœuds d'une structure 3D afin que la longueur des 18 arêtes satisfasse les spécifications spécifiées par l'utilisateur dimensions. La figure 5 illustre le Double-Banane dans sa configuration initiale.

La configuration de Double-Banana ne contient qu'un seul composant connexe, tel que révélé par BFS. L'analyse structurale utilisant la décomposition de DM montre qu'elle est sous-contrainte et notre algorithme suit alors la partie inférieure de la figure 1. L'analyse est utilisée dans notre fonction numFindRC et une surcontrainte est détectée. Plus spécifiquement, l'équation e9 est ici détectée. En utilisant notre approche de résolution incrémentale, l'équation est en outre caractérisée comme contradictoire. L'équation e9 est donc supprimée et le système est résolu en utilisant la position initiale des nœuds comme valeurs initiales des variables. En utilisant les résultats, l'équation e9 est alors réévaluée et le paramètre associé est comparé à la valeur spécifiée par l'utilisateur. Dans le cas présent, e9

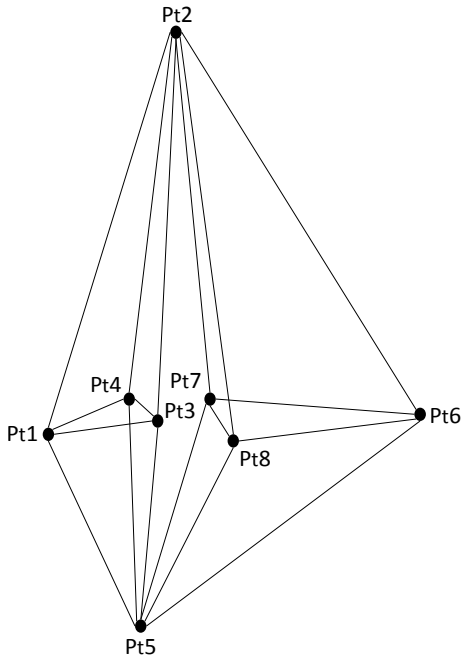


Figure 5: Géométrie initiale de la double banane comme décrit dans (Moinet, Mandil, and Serre, 2014).

n'est pas satisfait puisqu'il est égal à 44.47 par rapport à la condition initiale de 45 spécifiée par l'utilisateur. Ainsi, l'écart par rapport à l'intention de conception est de $df = 0.53/45$.

Notre algorithme donne une solution beaucoup plus proche de l'intention de conception initiale que l'algorithme de Moinet et al., et le système restant est moins mal-conditionné après la suppression de la contrainte conflictuelle (table 1). En fait, l'algorithme de Moinet et al. identifie e18 comme une équation conflictuelle et son retrait induit une déviation $df = 4.38/32$ de l'intention initiale de conception.

Méthode	Witness	Sur-contrainte	*df*	*cond*
Notre	esquisse initiale	e9	0.53/45	16.97
Moinet et al.	esquisse initiale	e18	4.38/32	73.33

Table 1: Comparaison entre notre algorithme et l'approche de Moinet sur le cas du test Double-Banane.

0.4.2 Esquisser un verre 3D

Dans cet exemple, l'idée est de montrer comment l'approche proposée de détection et de résolution de sur-contraintes peut supporter l'esquisse d'un verre 3D composé de 4 patchs NURBS connectés. Le concepteur esquisse son intention de conception et les exigences associées. L'objectif est de modifier

Contrainte	Équations	Type	Composant	Contrainte	Équations	Type	Composant
4	1-3	linéaire	1	12	30-32	non-linéaire	2
2	4-6	linéaire	2	13	33-35	linéaire	2
1	7-9	linéaire	1	14	36-38	non-linéaire	2
3	10-12	linéaire	2	15	39-41	linéaire	1
5	13	non-linéaire	1	16	42-44	non-linéaire	1
6	14	non-linéaire	2	17	45-47	linéaire	1
7	15-17	linéaire	1	18	48-50	non-linéaire	1
8	18-20	non-linéaire	1	19	51-53	linéaire	2
9	21-23	linéaire	1	20	54-56	non-linéaire	2
10	24-26	non-linéaire	1	21	57-59	linéaire	2
11	27-29	linéaire	2	22	60-62	non-linéaire	2

Table 2: Typologie des contraintes et des équations impliquées dans la description de l'exemple d'esquisse en verre 3D.

la partie supérieure du verre en spécifiant les éléments suivants:

- *Variables*: Chaque patch a un degré 5×5 et a un polygone de contrôle fait de 16×6 points de contrôle dont les coordonnées sont les variables de notre processus d'optimisation (figure 6.a). Puisque l'objectif est de modifier la partie supérieure du verre, le concepteur choisit combien de rangées de points de contrôle doivent être bloquées et combien peuvent bouger. Par exemple, si le concepteur souhaite libérer la rangée supérieure de points de contrôle du quatre patches, il y aura alors $6 \times 4 \times 3 = 72$ variables dans le vecteur inconnu X. Les résultats seront illustrés avec 4 et 5 lignes libres de se déplacer.

- *Contraintes*: Trois types de contraintes sont utilisés pour spécifier comment la forme du verre 3D doit évoluer:

  - *Position*: 4 contraintes de position sont ajoutées aux quatre extrémités des patches le long des courbes de limite supérieure. Comme le montre la figure 6.c, les points verts des patches doivent se déplacer vers de nouvelles positions dans Espace 3D. Ils sont étiquetés de 1 à 4 et génèrent $4 \times 3 = 12$ équations linéaires étiquetées de 1 à 12 (table 2).
  - *Distance*: 2 contraintes de distance sont définies entre les côtés opposés des patchs (figure 6.d). Ils sont étiquetés 5 et 6 et ils génèrent 2×1 équations non linéaires étiquetées 13 et 14 (table 2).
  - *Coincidence and tangency*: 8 coïncidences et 8 contraintes de tangence sont spécifiées pour maintenir la continuité entre les parties supérieures des patches pendant la déformation (figure 6.b). Ils sont étiquetés de 7 à 22 et ils génèrent des équations non linéaires de 8×3 linéaire et de 8×3 étiquetées de 15 à 62 (table 2). La non-linéarité provient de l'utilisation du produit vectoriel pour exprimer la colinéarité des normales.

Dans l'ensemble, il existe 22 contraintes géométriques générant 62 équations dans l'ensemble $F(X) = 0$. Certaines de ces contraintes sont contradictoires et c'est le but de cette section d'essayer de voir comment notre algorithme peut les détecter et les supprimer sans affectant trop l'intention de conception.

- *Objective function*: Puisque l'approche proposée supprime les sur-contr aintes identifiées, le système d'équations résultant $\widetilde{F}_b(X) = 0$ (section 0.3.4) peut devenir sous-contraint et un la fonction $G(X)$ doit être minimisée. Ici, l'idée est de faire usage de l'approche de Pernot et al. pour définir deux types de comportement de déformation (Pernot et al., 2005): soit une minimisation de la variation de la forme ($G_1(X)$) entre les configurations initiale et finale, soit la minimisation de la forme finale ($G_2(X)$). En termes d'intention de conception, le premier a tendance à conserver la forme initiale du verre, alors que le second oublie la forme initiale et tend à générer des surfaces similaires aux structures de traction.

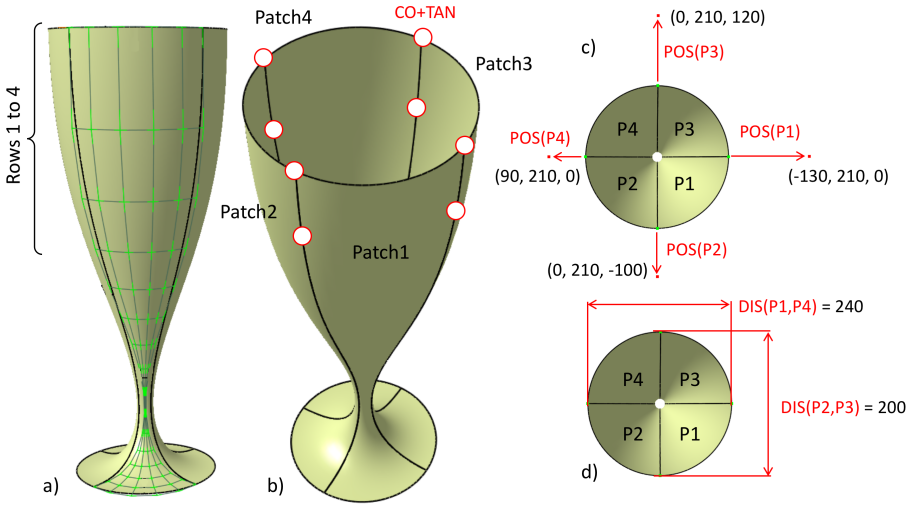


Figure 6: Esquisse initiale de la géométrie du verre

Comme l'a révélé BFS, le système peut être décomposé en $2+24\times \mathrm{N}_{rows}$ connection CC_i où N_{rows} est le nombre de lignes libres de bouge toi. Parmi eux, seuls deux composants CC_1 et CC_2 contiennent à la fois des variables et des équations alors que les autres ne contiennent que des variables (Table 2). L'analyse de ces deux composantes donne lieu à l'identification de 2 équations conflictuelles qui correspondent soit à la position, soit à la distance. Le résultat du processus de détection n'étant pas unique, 9 configurations sont obtenues et sont rassemblées dans le table 3. Ici, il faut se rappeler

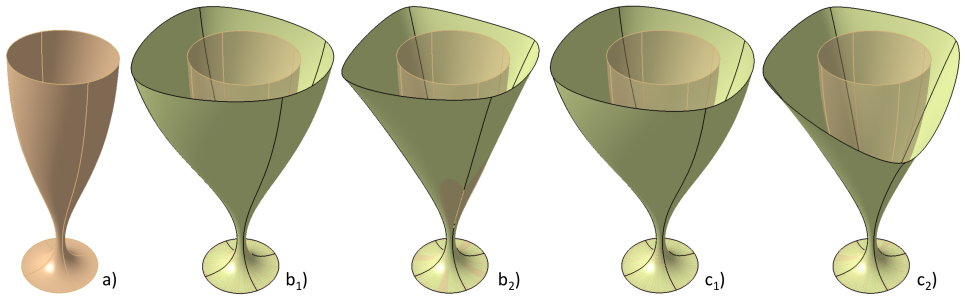


Figure 7: Résultats de l'esquisse après suppression des contraintes conflictuelles avec $N_{rows} = 4$: (a) verre initial, (b_1) la configuration 1 et minimisation de la variation de forme, (b_2) la configuration 1 et la minimisation de la surface de la surface finale, (c_1) la configuration 3 et minimisation de la variation de forme, (b_2) la configuration 3 et la minimisation de la surface de la surface finale.

que même si le processus de détection identifie des équations conflictuelles, notre algorithme supprime les contraintes associées à ces équations. Par exemple, la configuration 1 considère que les deux contraintes de distance (une entre les patches P1 et P4 et l'autre entre P2 et P3) doivent être supprimées (0 dans la table) et les 4 contraintes de position sont conservées (1 dans la table).

Config.	DIS(P1,P4)	DIS(P2,P3)	POS(P1)	POS(P2)	POS(P3)	POS(P4)
1	0	0	1	1	1	1
2	1	0	0	1	1	1
3	1	0	1	1	1	0
4	0	1	1	0	1	1
5	0	1	1	1	0	1
6	1	1	0	0	1	1
7	1	1	1	1	0	0
8	1	1	1	0	1	0
9	1	1	0	1	0	1

Table 3: Statut des contraintes de distance et de position (0 à enlever et 1 à garder) pour résoudre les 9 configurations sur-contraintes.

Toutes les configurations sont alors résolues en agissant à la fois sur le nombre de rangées supérieures à fixer ($N_{rows} = 4$ or 5), et la fonction objectif à minimiser (soit $G_1(X)$ ou $G_2(X)$). Les résultats sont rassemblés dans les tableaux 4 et 5. Chaque configuration est évaluée à l'aide des trois critères précédemment introduits *dg*, *df* et *cond*. Certaines solutions sont montrées dans la figure 7.

On peut d'abord remarquer que selon les configurations, l'écart *df* sur les contraintes varie. Par exemple, avec $N_{rows} = 4$ et en minimisant $G_1(X)$, la configuration 7 génère une solution plus proche de l'intention de conception

que la configuration 6 ($0.10684 < 0.12607$ dans Table 4). Pour la configuration 3, il est clair que l'écart par rapport à l'intention de conception en termes de contraintes est plus important en minimisant la surface de la surface finale qu'en minimisant la variation de forme ($0.2288 > 0.10179$ dans le table 4). Ceci est clairement visible sur les figures 7.c$_1$ et 7.c$_2$.

En considérant la minimisation de la variation de forme, on peut voir que la configuration 3 est moins intéressante que la configuration 1 en ce sens qu'elle minimise moins la variation de forme ($15459.52 > 13801.04$ dans le table 4).

Enfin, pour une configuration donnée, on peut remarquer que lorsque le nombre de lignes libres augmente, c'est-à-dire quand il y a plus de liberté, la fonction objective diminue et la solution est donc plus proche de l'intention de conception. Ceci est visible lors de la comparaison des valeurs des Tableaux 4 et 5. Ainsi, la sélection des variables X est également importante lors de la mise en place du problème d'optimisation.

	Minimization of $G_1(X)$			**Minimization of** $G_2(X)$		
Config.	dg_1	df	$cond$	dg_2	df	$cond$
1	13801.04	0.10000	2.8654e19	96733.72	0.10000	1.4272e18
2	17990.88	0.10182	8.3172e18	95225.05	0.28157	4.3894e17
3	15459.52	0.10179	1.5071e19	94483.08	0.22880	4.9533e17
4	12265.51	0.10975	8.8857e18	89924.13	0.22806	3.9399e18
5	10970.98	0.10971	3.9852e19	86879.47	0.25225	8.2501e18
6	15826.68	0.12607	3.4260e18	76878.26	0.68278	1.6567e18
7	12936.45	0.10465	3.8205e18	76167.99	0.62820	3.9842e17
8	13889.18	0.11385	2.5681e18	78657.81	0.59160	4.1485e18
9	14883.21	0.11720	1.2523e18	74351.81	0.71765	6.7658e16

Table 4: Evaluation des 9 configurations avec $N_{rows} = 4$.

	Minimization of $G_1(X)$			**Minimization of** $G_2(X)$		
Config.	dg_1	df	$cond$	dg_2	df	$cond$
1	11266.93	0.10000	4.0149e17	85121.36	0.10000	3.3441e17
2	14719.05	0.10280	4.6031e17	86295.47	0.25034	6.5355e19
3	12506.55	0.10277	1.7748e19	85190.96	0.20076	1.0972e18
4	9944.87	0.11452	1.7903e18	79428.31	0.20592	1.7041e18
5	8799.29	0.11448	6.1454e17	77800.57	0.22919	1.0218e18
6	12561.66	0.13935	4.1681e18	69603.16	0.76646	8.5100e16
7	10441.11	0.10684	1.0862e18	69502.72	0.70009	2.3460e18
8	11134.09	0.12097	2.5394e18	71465.72	0.65901	1.5773e18
9	11877.59	0.12601	1.3790e19	67661.55	0.80372	8.4472e17

Table 5: Evaluation des 9 configurations avec $N_{rows} = 5$.

0.5 Conclusion et travaux futurs

Dans ce travail, une approche pour trouver toutes les sur-contraintes dans des configurations géométriques de forme libre a été introduit. Il s'appuie sur un couplage entre les décompositions structurelles et l'analyse numérique. Le processus et son algorithme ont été décrits et analysés avec des résultats à la fois exemples académiques et industriels. L'approche a plusieurs avantages: elle est capable de distinguer les contraintes redondantes et conflictuelles; il est applicable à la fois sur les contraintes linéaires et non linéaires; et cela applique des méthodes numériques sur de petits sous-blocs du système original, permettant ainsi d'évoluer vers de grandes configurations. De plus, puisque l'ensemble des sur-contraintes d'un système n'est pas unique, il a été montré que notre approche est en mesure de fournir différents ensembles en fonction de la la décomposition structurelle sélectionnée et les critères proposés pour comparer et aider l'utilisateur à choisir les contraintes il/elle veut supprimer. Même si le noyau de l'algorithme fonctionne sur des équations et des variables, la décision est prise en considérant les contraintes géométriques spécifiées par le concepteur à un niveau élevé.

Un certain nombre de perspectives découlent de ce travail. D'abord, une automatisation de processus devrait aider le concepteur à choisir l'ensemble. Le concepteur a accès à trois critères principaux (dg, df, $cond$) qui peuvent être difficiles à analyser pour un non-expert. Ainsi, des critères de niveau supérieur devraient être imaginés en plus de ceux-ci. Deuxièmement, l'approche peut être rendue interactive, c'est-à-dire permettre le concepteur de choisir entre les différents ensembles en conflit le long de la processus, ou même de modifier les contraintes défectueuses. Enfin, il est prévu de étendre ce travail afin qu'il puisse être utilisé pour détecter et expliquer configurations géométriques qui, même lorsqu'elles sont solubles, se traduisent des conceptions de qualité.

Introduction

Nowadays, designers rely on 3D CAD software to model sophisticated shapes based on free-form curves and surfaces. In industrial design, this geometric modeling step is often encapsulated in a larger Product Development Process (PDP) which may incorporate preliminary design, reverse engineering, simulation as well as manufacturing steps wherein several actors interact. Actually, the final shape of a product often results from a long optimization process which tries to satisfy the requirements associated to the different steps of the PDP. Requirements can be seen as constraints. They are generally expressed either with equations, a function to be minimized, and/or using procedures.

To satisfy the requirements, designers can act on variables associated to the different steps of the PDP. More specifically, variables are supposed to be the parameters of the NURBS surfaces involved in the shape optimization process. To shape a free-form object defined by such surfaces, designers then have to specify the geometric constraints the object has to satisfy. For example, a patch has to go through a set of 3D points and satisfy to position constraints, the distance between two points located on a patch is fixed, two patches have to meet tangency constraints or higher-order continuity conditions, etc. Those geometric constraints give rise to a set of linear and non-linear equations linking the variables whose values have to be found. Due to the local support property of NURBS, the equations do not involve all the variables and some decompositions can be foreseen. Additionally, designers may express involuntarily several times the same requirements using different constraints thus leading to redundant equations. But the designers may also involuntarily generate conflicting equations and may have to face over-constrained and unsatisfiable configurations.

Sometimes, over-constrained configurations can be solved by inserting extra degrees of freedom (DoFs) with the Boehm's knot insertion algorithm. As a consequence, many control points are added in areas where not so many DoFs are necessary. This uncontrolled increase of the DoFs impacts

the overall quality of the final surfaces which become more difficult to manipulate than the initial ones. Furthermore, some structural over-constraints cannot disappear following this strategy and dedicated decision-support approaches have to be developed to identify and manage over-constrained configurations.

Unlike advanced 2D sketchers available in most commercial CAD software, and which can interactively identify the over-constraints during the sketching process, it is not yet completely possible to pre-analyze the status of 3D NURBS-based equation systems before submitting them to a solver. Thus there is a need for developing a new approach for the detection and resolution of redundant and conflicting constraints in NURBS-based equation systems. This corresponds to the identification and treatment of over-constrained, well-constrained and under-constrained parts. In this thesis, the treatment corresponds to the removal of constraints before solving. Once the constraints removed, the equation system often becomes under-constrained and the designer also has to add a requirement by mean of a function to be minimized so as to solve and find the values of the unknowns. This aspect is not part of the proposed approach but it will be discussed when introducing the results in which a particular functional is minimized.

Removing user-specified constraints is a primary step as the result do not fully satisfies what the designers have specified. Thus, not only is it important to develop an approach able to remove over-constraints, but it is also desirable to develop decision-support mechanisms which can help the designers identifying and removing the right constraints, i.e. the ones which preserve as much as possible the initial design intent.

In this thesis, the aim is to address these two difficult issues by proposing an original decision-support approach to manage over-constrained geometric configurations when deforming free-form surfaces. Our approach handles linear as well as non-linear equations and exploits the local support property of NURBS. Based on a series of structural decompositions coupled with numerical analyses, the method detects and treats redundant as well as conflicting constraints. Since the result of this detection process is not unique, several criteria are introduced to drive the designer in identifying which constraints should be removed to minimize the impact on his/her original design intent. Thus, even if the kernel of the algorithm works on equations and variables, the decision is taken by considering the geometric constraints specified by the user at a high level.

The manuscript is composed of an introduction, 4 chapters and a final conclusion as follows:

- **Chapter 1** shows the whole picture of Product Development Process (PDP) and points out the position of our research. We introduce the structure of PDP, its relationship with free-form shape modeling, modeling user specified requirements, and transforming from requirements to constraints, from constrains to equations.

- **Chapter 2** provides an overview of techniques for geometric over-constraints detection. In the context of free-form surfaces modeling, we evaluate different techniques under given criteria and select the ones that might be helpful to solve our problems by testing them on different examples.

- **Chapter 3** illustrates our approach: an algorithm combining system decomposition, numerical methods, symbolic methods, and optimization techniques to detect redundant/conflicting constraints as well as the corresponding spanning groups. In addition, our approach enables to provide different sets of results and thus criteria are proposed to compare them, to assist user choosing the over-constraints he/she wants to remove.

- **Chapter 4** illustrates the detection and resolution processes on both academic and industrial examples. Moreover, it shows the efficiency of the decomposition method used in our approach and the impact of tolerances on the detection result.

- Finally, we discuss limits and perspectives of our research in the **Conclusions and perspectives** section.

Chapter 1

Positioning of the research

This chapter discusses briefly the Product Development Process (PDP) and shows that its output can be seen as the result of an optimization problem where multiple requirements should be satisfied (section 1.1). Requirements can be realized by manipulating geometric models in CAD modelers using different approaches (section 1.2). Modeling multiple requirements on freeform objects can be seen as defining a shape deformation problem (section 1.3). Requirements can be transformed into constraints (section 1.4) and constraints can be further transformed into equations (section 1.5). Since the users' design intent are sometimes uncertain and requirements may contain redundancies/conflicts, there is a need to debug them so that users can better understand what they really want (section 1.6).

1.1 Product Development Process

Product design is a cyclic and iterative process, a kind of systematic problem solving, which manages the creation of the product itself under different conditions. The development process includes the idea generation, concept phase, product styling and design and detail engineering, all of which are conducted in the context of adapting and satisfying requirements of the different stages. Usually, designers use CAD tools to deal with associated requirements during the industrial Product Development Process (PDP). Clearly, the PDP is not unique and vary deeply from company to company, depending on the complexity of the product to be designed, the equipment used, the team of specialists involved, the stages of a product to be designed and so on. Regardless of the variability, Falcidieno at al (Falcidieno et al., 2014) proposed a generic structure of a PDP as a reference scheme that can be used by specific companies, on different scenarios and classes of products (figure 1.1). Singe-headed arrows represent information

and/or digital models that are communicated from one activity to another as soon as the first activity has been carried out. Double-headed arrows are not prescribing systematic communications between two or more activities. They can be reduced to single-way communications for some specific scenarios (Falcidieno et al., 2014).

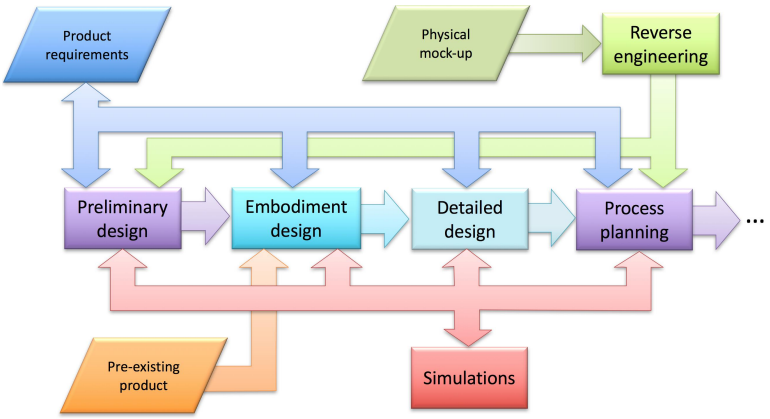


Figure 1.1: Generic structure of a PDP.

As it is shown in figure 1.1, the PDP can be divided into four main stages: Preliminary design, Embodiment design, Detailed design and Process planning. In the Preliminary design stage, the desired product characteristics are described and the product requirements are defined. The description of product characteristics provides the basis for the definition of the requirement specifications and target specifications of a new model. The requirement specifications include a complete description of the new product characteristics. For example, in the context of automotive development projects, the description of product characteristics is supported by far-reaching market studies, research into constantly changing customer demands and on evaluation of future legislation-based conditions in target markets. Requirement specifications takes into account the detailed information about the requirements of product design and the desired behavior of a product in terms of its operation. The Embodiment design stage includes the functional and physical concept of the new product. On one hand, the new techniques are used and evaluated with respect to their functional configurations and interactions. For example, in the case of car design, new technologies in mechanical parts, such as new safety equipment or environmentally friendly propulsion technologies, are implemented and verified in terms of their general functionalities within the full-vehicle system. On the other hand, the physical concept covers the definition of the product composition. New functions are

developed in mechanisms. In automotive development processes, the physical concept defines the vehicle body structure layout in consideration of crash and stiffness requirements as well as it addresses basic requirements of the new car concept, such as driving performances, fuel consumption, vehicle mass, and estimated values of driving dynamics.

The Detailed design stage directly depends on the product concept phase. Based on the knowledge from the concept phase, the geometric information of all components are modeled in details and optimized while considering the assembly of the product and the interactions of components. Also, in this stage, materials of the components are defined and the boundaries for the production planning are derived.

Finally, the last stage consists of production-related planning. This stage is mainly concerned with determining the sequence of individual manufacturing operations needed to produce a given part or product. But it also refers to the planning of use of blanks, spare parts, packaging material, user instructions, etc. The resulting operation sequence is documented on a form typically referred to as a route sheet containing a listing of the production operations and associated machine tools for a work part or assembly. This phase goes hand in hand with the design process because manufacturing boundaries often influence the design of components. Therefore, the production, assembly and inspection-oriented development and the manufacturing-related optimization interact with geometry creation and calculation processes.

With the goal of saving development time and costs, the PDP involves virtual product-model-based processes to generate new products using dedicated tools. Depending on the categories of development applied, there are different types of tools: computer-aided design (CAD), computer-aided styling (CAS), computer-aided engineering (CAE), Computer-aided manufacturing (CAM), Computer-aided quality assurance (CAQ) and Computer-aided testing (CAT). These tools enable the creation of product geometry and implementation of product characteristics. All those information are stored within the Digital Mock Up which is managed by PDM (Product Design Management) and PLM (Product Lifecycle Management) systems. Modern CAD systems allows for integrating multi-representation and multi-resolution geometric models to shape complex components and modeling products possibly incorporating free-form surfaces (Pernot et al., 2008). The CAD modeling approaches will be discussed in the next section.

Anyhow, the PDP allows the creation of products which can be seen as the result of an optimization process where various requirements (e.g. functional, aesthetic, economical, feasibility) have to be satisfied so as to obtain desirable solutions. However, not all the requirements can be fulfilled

and an approximation has to be found. Moreover, requirements are even conflicting in some cases and methods for detecting and treating conflicts are to be proposed. They are presented and compared in chapter 2. This PhD thesis addresses such a difficult problem of detection and treatment of so called over-constraints when manipulating geometric models which are part of the DMU of complex systems.

1.2 CAD modeling approaches

1.2.1 Manipulating geometric models

As suggested by Maculet and Daniel in (Maculet and Daniel, 2004), the manipulation of a geometric model can be performed at three levels:

- Level 0: manipulation of variables, or parameters (e.g.: coordinates of a control point, coordinates of a point in parametric space...).
- Level 1: manipulation of elementary geometric entities (points, line segments, curves, surfaces); it corresponds to the parametric and variational modelers solving elementary geometric constraints (e.g.: distance between two points, angle between two tangent lines, etc.).
- Level 2: manipulation of more complex geometric entities, composed of simple elements of level 1(e.g.: groove in an area of an object); it corresponds to the feature-based approaches, to solve more complex constraints (e.g.: length of the groove), and which are generally associated with a semantic meaning or with geometric properties.

Industrial CAD software relies on an incremental B-Rep (Boundary Representation) modeling paradigm where volume modeling is performed iteratively through high-level operators (Hoffmann, Lomonosov, and Sitharam, 1998). These operators allows for acting directly on the geometric entities of level 1 to directly shape the CAD models by manipulating structural and detail features. However, even if CAD software allow working on the geometric entities of level 2 based on the operators such as pad, pocket, shaft to get rid of the direct use and manipulation of canonical surfaces and NURBS (Piegl and Tiller, 1996), a lot of intermediate operations are required to get the desirable shape of an object. The work is procedural and designers have to break down the object body into basic shapes so as to link to different operators of the software. This even truer in the freeform domain where CAD softwares generate complex free-form shapes incrementally and interactively through a sequence of simple shape modeling operations. The chronology of

these operations is at the basis of a history tree describing the construction process of an object. Consequently, without a real construction tree, free-form shape modifications are generally tedious and frequently result in update failures. Clearly, an approach closer to the designers' way of thinking is missing and there is still a gap between the shapes designers have in mind and the tools and operators provided to model them. Various approaches have been introduced to bridge this gap and are briefly discussed in the next section: parametric modeling, feature-based modeling and variational modeling approaches.

1.2.2 Modeling approaches

The parametric modeling approach allows for modifying an object by changing instantiations of its constitutive geometric model sequentially. Usually, a parametric system can be divided into subsystems that can be solved one after another in a given order (Farin, Hoschek, and Kim, 2002). During the process of parametric modeling, designers are hard to make sure that the added constraints stay consistent with previous ones as well as the inserted variables are enough to correctly describe the variations of a product. Therefore, final systems are generally under-constrained or sometimes over-constrained. Designers have to check carefully the constraints with respect to the product variations he/she needs so that the final system stays well-constrained.

However, if a system can be divided into subsystems which are simultaneously solvable, then the system is variational. Designation of variational configuration is first proposed by Lin, Gossard and Light (Lin, Gossard, and Light, 1981). When defining a 3D shape, the constraints are often geometric constraints, which relate to different geometric primitives or features (Bettig and Hoffmann, 2011). For example, they can be distances or angles between (special points or axes of) geometric primitives or features, incidence or tangency relations between parts of two geometric primitives or features. Compared to the parametric modeling, variational modeling gives a better answer to the designer's needs. If the final configuration is well-constrained, the solver is able to find a correct set of solutions. In configurations that are under/over-constrained, decomposition methods have been introduced to supplement the solving methods. The research in variational modeling quickly concentrates on the important problem of constraints modeling, how to represent and organize them with DAGs, and finally how to solve them (Hoffmann and Juan, 1992).

The feature-based modeling approach considers geometric entities made

up of simple elements which are called features. For example, the feature 'hole' is composed by a set of cylinders and planes attached to an initial plane (figure 1.2). In the freeform domain, four level classification has been proposed by Fontana (Fontana, Giannini, and Meirana, 2000) and extended by Pernot (Pernot, 2004). Designers manipulate directly on shape primitives that can be parametrized and pre-defined rather than acting at lower level ones by using features to build their CAD models. As it is shown in figure 1.3, constraints can be indirectly specified on the control points of control polygon or directly onto the surface potentially made of multiple trimmed patches connected together with continuity conditions. To satisfy their requirements, designers often insert numerous DOFs through the use of the Boehm's knot insertion algorithm (Boehm and Prautzsch, 1985), resulting in configurations with more variables than equations. The approach proposed

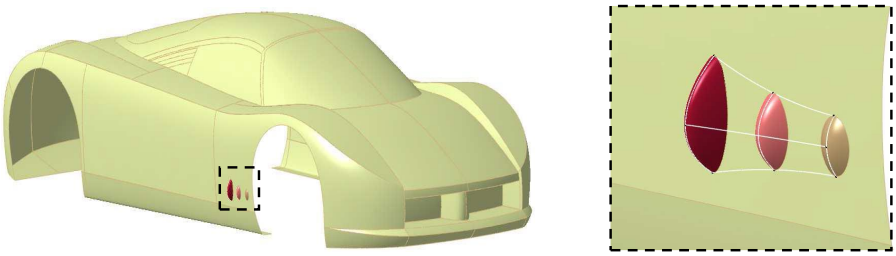


Figure 1.2: Geometric model composed of free form features (Pernot et al., 2005).

in this PhD can serve the three above mentioned modeling strategies. As discussed in section 1.3, the proposed approach is intended to analyze a set of constraints, and the associated equations, independently of the adopted modeling strategy.

1.3 Modeling multiple requirements in an optimization problem

Product requirements refer to the specifications that lead to criteria to evaluate design variants and select the one that performs best when using the product. As it is shown in Figure 1.1, requirements can be specified during various stages of a PDP from preliminary design to process planing. Usually, the requirements are of two categories: qualitative and quantitative ones. Quantitative requirements like a power or a velocity can be subjected to tolerances, which gives a flexibility to find the compromise. However, qualitative requirements like aesthetics are not related to tolerances, and

compromises are more subjective, which in some sense can reduce the uncertainties. To find a compromise, a set of requirements can be adjusted like adding/removing shape details, modifying dimensions, applying geometric constraints on the digital shape models. Often, the product shape results from an optimization problem where the various requirements are specified by the actors of the PDP. Actually, the final shape of a product often results from a long and tedious optimization process which tries to satisfy the requirements associated to the different steps and actors of the PDP. Those requirements can be of different types and their computation may require the need of external tools or libraries. For example, the shape of a turbine blade is the result of a complex optimization process which is to find the best compromise between notably its aerodynamic and mechanical performances. In general, requirements can be seen as constraints. They are generally expressed either with equations, a function to be minimized, and/or using procedures (Gouaty et al., 2016). The latter refers to the notion of black box constraints, which are not addressed in this manuscript (section 1.5.2). Here, we focuses only on **geometric constraints that can be expressed by linear or non-linear equations**.

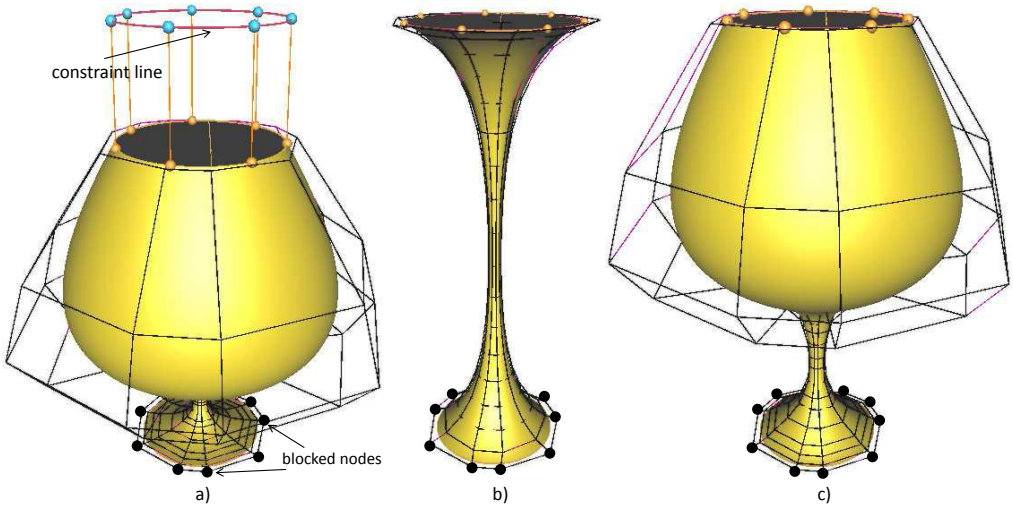


Figure 1.3: Initial configuration a), deformation with a minimization which ignores the initial shape b), and preserves the initial shape c) of initial glass (Pernot et al., 2005).

Thinking to the PDP as well as to the needs for generating shapes which satisfy multiple requirements, one can notice that designers can specify their requirements and associated design intent within a shape deformation problem through accessing to three main parameters. On one hand, when speak-

ing of deformation techniques working on NURBS curves and surfaces, the goal is to find the position X of some control points so as to satisfy user-specified constraints which can be translated into a set of linear and/or non-linear equations $F(X) = 0$. For example, a patch has to go through a set of 3D points and satisfy to position constraints (some of these constraints have been used to drive the glass deformation of figure 1.3), the distance between two points located on a patch is fixed, two patches have to meet tangency constraints or higher-order continuity conditions, etc. Since the problem is often globally under-constrained, i.e. there are less equations than unknown variables, an objective function $G(X)$ also has to be minimized. As a consequence, the deformation of free-form shapes often results from the resolution of an optimization problem:

$$\begin{cases} F(X) = 0 \\ \min G(X) \end{cases} \tag{1.1}$$

For some particular applications, the optimization problem can also consider that the degrees, the knot sequences or the weights of the NURBS are unknown. Depending on the approach, different objective functions $G(X)$ can be adopted but they often look like an energy function which may rely on mechanical or physical models. Figure 1.3 b and c show two results when using two different minimization with the same set of constraints. The constraints toolbox can also contain more or less sophisticated constraints with more or less intuitive mechanisms to specify them. Note that, in this thesis, **only the position of the control points are considered unknown**. On the other hand, designers can effectively act on the unknowns X to decide which control points are fixed and which ones can move. In the example of figure 1.3, the bottom row of control points are fixed. In this way, they specify the parts of the initial shape which should not be affected by the deformation. Of course, designers can make use of the constraints toolbox to specify the equations $F(X) = 0$ to be satisfied. Finally, designers can also specify some of their requirements through the function $G(X)$ to be minimized. For example, they can decide to preserve or not the original shape while minimizing an energy function characterizing the shape deformation.

1.4 From requirements to constraints

As discussed above, multiple requirements in a PDP can be modeled within an optimization problem. Here, we summarize the constraints satisfying different design requirements as well as a structure of these constraints which must be incorporated to develop a fully constraint-based modeler for curves and surfaces.

1.4.1 Taxonomy of constraints

Section 1.3 has discussed the design intent where both the system of equations $F(X) = 0$ and the objective function $G(X)$ should be taken into consideration. Therefore, the corresponding ways of specifying constraints can be (Cheutet et al., 2007):

- Strict constraints, which are named classically constraints in the literature and can be transformed into a system of equations $F(X) = 0$. They must be strictly respected during the shape creation and manipulation processes. For example, the current sketchers in CAD modelers are only using this type of constraints.
- Soft constraints, which corresponds to objective function $G(X)$ to be minimized. These constraints are used in the declarative modeling approach to allow the description of the object properties, but also to deform free-form surfaces in some other approaches. They can express the final aspect of a component shape or at least, the expectation to obtain a solution close to it.

The above two categories classify constraints specification from a mathematical point of view. However, thinking to the PDP, the notion of constraints during a design phase can be very large. Since it is commonly used at all of its successive steps, different users have different meaning for design constraints. According to (Cheutet et al., 2007), constraints can be classified into four semantic levels in the context of shape generation and modification, depending on the type of the constrained entity:

- Level 1: constraints attached to a geometric element of a configuration: such as local constraints used to manipulate its shape like position constraints.
- Level 2: constraints between two or more geometric elements of a configuration: for instance, to preserve the integrity of the configuration during the shape modification, such as maintaining G0/G1/G2 continuity between trimmed patches.
- Level 3: constraints attached to the whole configuration like a volume constraint, for example.
- Level 4: constraints related to the product itself rather than to the geometry. For example, the product should resist during its usage. This specification makes use of mechanical properties such as the acceptable maximum stress. In this case, constraints link the geometry with

parameters of the material as well as boundary conditions of the product.

The above levels describe how to express the constraints attached to a product. In the next subsections, the constraints classically used for curve and surface modeling are described in more details, according to the categories previously defined in this section. Most of those constraints can be handled by the approach developed in this PhD thesis.

1.4.2 Strict constraints

This section describes the strict constraints commonly used in shape modeling and part of the first two categories previously listed (constraint attached to one geometric element or between two or more geometric elements of a component). Because they are directly related to geometric parameters, they are named as geometric constraints in literature.

Constraints to control the shape of a local entity

Local geometric constraints are used to locally control a shape. The control is achieved through enforcing the curve/surface to pass through a new user-specified location using the local support property of the underlying geometric model (figure 1.4, b). The constraining entity usually is a geometric point while the constrained geometry can be curves, patches and meshes.

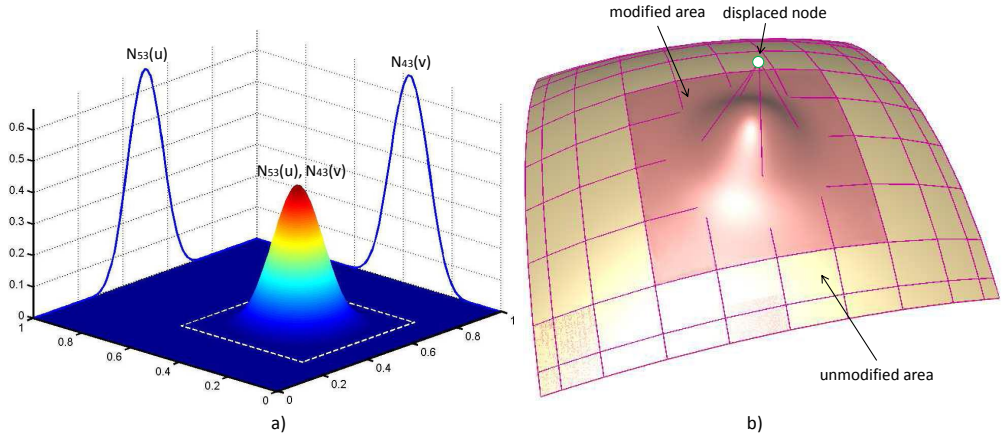


Figure 1.4: Bi-variate basis function a) associated to a control point which is displaced b) to produce a local modification of a patch (Pernot et al., 2005)

The local support property can be explained as following. A B-Spline

patch is defined by the following equation:

$$\begin{array}{c}(u,v)\in \mathcal{N}_u\times \mathcal{N}_v, P(u,v)=\sum_{i=0}^{m}\sum_{j=0}^{n} N_{ip}(u).N_{jq}(v).s_{ij},\\ \text{with } \mathcal{N}_u=[u_0,u_{m+p+1}] \text{ and } \mathcal{N}_v=[v_0,v_{n+q+1}],\end{array} \tag{1.2}$$

where p and q are the degrees in u and v respectively, $m+1$ and $n+1$ are the number of control points in u and v direction respectively, $\mathcal{N}_u$ and $\mathcal{N}_v$ are the knot sequences in u and v.

The displacement $\underline{\boldsymbol{\delta}}_{hk}$ of a control point s_{hk} induces a surface displacement governed by the following equation:

$$(u,v)\in \mathcal{N}_u\times \mathcal{N}_v, P^{'}(u,v)=P(u,v)+N_{hp}(u).N_{kq}(v).\underline{\boldsymbol{\delta}}_{hk} \tag{1.3}$$

Thus, the extent of the modified area directly depends on the **influence area** of the bi-variate basis function $N_{hp}(u).N_{kq}(v)$, whereas the amplitude of the modification is directly related both to the shape of this bi-variate function and to the displacement vector $\underline{\boldsymbol{\delta}}_{hk}$. The example of figure 1.4 a, displays the bivariate basis function associated to a control point which is displaced to produce a local modification of a patch (figure 1.4, b). The influence area of this displacement is therefore delimited by a rectangular domain $\mathcal{I}_{hk}=[u_h,u_{h+m+1}]\times[v_k,v_{k+n+1}]$. This is an interesting property to be able to decompose a problem into subproblems.

Curve constraints on a surface

In car aesthetic design, stylists manipulate a product by forcing the surface to match a given curve at some stage of the product specification. Thus, the surface model of the product is directed by the curves, which strongly affect the shape of the product. The constraining entity is a curve and the constrained entity is a surface. In the case of continuous surfaces like NURBS surfaces, the curve constraint can be decomposed into a set of point constraints, with additional parameters related to the application domain (figure 1.5). The discretization process has a strong influence on the resulting shape. For example, if the discretization is too coarse, then the shape variations of the initial curve would be lost; If the degrees of freedom is not properly distributed after discretization, then over-constrained configurations will be generated. There are cases for which the decomposition is not necessary, but they are so particular that they are not often encountered in industry (Michalik and Brüderlin, 2004).

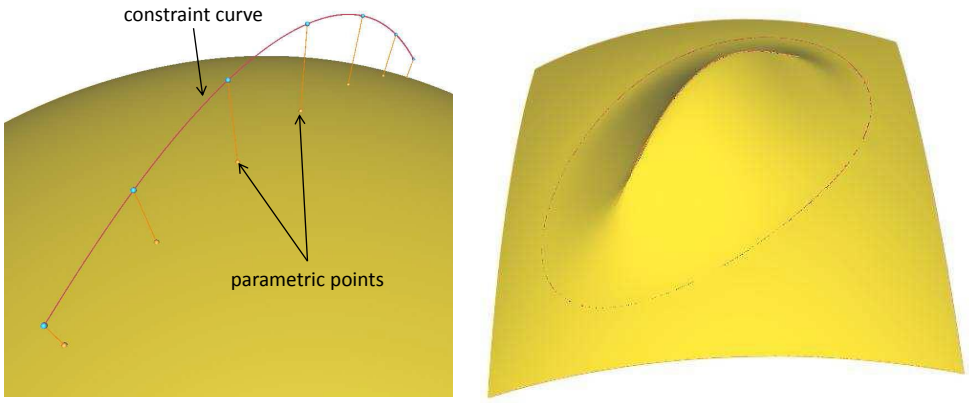


Figure 1.5: Matching a curve constraint a) initial configuration b) shape after deformation and insertion of a discontinuity (Pernot et al., 2005)

Constraints to preserve model integrity

Figure 1.5 b) shows a shape that is not "smooth" after deformation. This discontinuity results from the trimmed self-intersection of the surface. Thus, the integrity of the original configuration is not well preserved in this case. In case of continuity conditions, the shape of a model can be defined by a set of patches and the continuity conditions in position, tangency and/or curvature between them have to be taken into account to preserve the model integrity during the shape transformations (second semantic level, section 1.4.1). More specifically, for a surface composed of a set of connected trimmed patches, the continuity of the surface depends on the continuity of each patch and on the continuity along their connections. Concerning the connections, a C^0 continuity indicates a **continuity** of position along the common trimming lines of two patches, a C^1 indicates a continuity of the first-order derivatives along the common trimming lines and a C^2 continuity indicates a continuity of the second-order partial derivatives along the common trimming lines. Unfortunately, most of the time the characteristics of the two connected patches (degrees of the basis functions, number of control points and so on) prohibit the satisfaction of those equalities. Therefore, the continuity must be **approximated** (G^i instead of C^i) and satisfied at specific points of the trimming lines. In this case, the G^0 continuity corresponds to position continuity at specific points, G^1 to tangency continuity and G^2 to curvature continuity.

Constraints to control the shape of a global configuration

This section describes constraints that act on the whole curve/surface (constraints from semantic level 3). They cannot be decomposed into a set of point constraints like previous examples since they refer to integral properties of the associated curve/surface. In 2D space, curves can be constrained to preserve either a prescribed area or a constant length or to preserve some symmetry with a predefined axis during the deformation process (Sauvage, Hahmann, and Bonneau, 2004; Hahmann, Sauvage, and Bonneau, 2005; Elber, 2001). In 3D space, the volume preservation is important for achieving realistic deformations of solid objects in computer graphics (Lasseter, 1987). Other constraints such as moments have been studied in (Elber, 2000; Gonzalez-Ochoa, McCammon, and Peters, 1998). An example of area constraint is provided in figure 1.6.

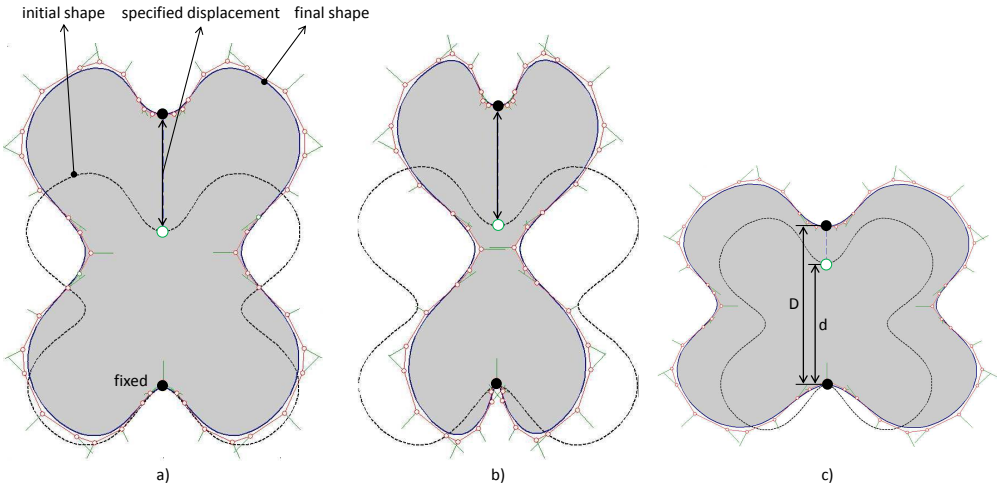


Figure 1.6: a) no area constraint b) constant inside area c) inside area scaled with a factor $\alpha = D^2/d^2$ (Pernot et al., 2005)

Constraints to satisfy engineering requirements

This section deals with constraints of the fourth semantic level. As discussed in section 1.4.1, these constraints are usually needed at a given stage of the PDP and their expressions incorporate geometric as well as technological parameters, such as changing the shape of a component in some areas while maintaining the maximum stress value in a given area. However, usually these constraints cannot be decomposed into constraints of the other semantic levels since other quantities than the geometric ones are in-

volved. Moreover, evaluating the results satisfying these constraints would require the use of a specific algorithm, like a Finite Element Analysis. Most of time, these constraints are seen by the designer as black boxes and the results obtained are then incorporated into a geometric constraint solving process (section 1.5.2). After that, the user can have ideas on which parameters need to be modified by analyzing the solutions. This level is not addressed in this PhD manuscript, and black box constraints cannot be handled yet.

1.4.3 Soft constraints

Constraints in terms of objective function to minimize

(Pernot et al., 2004) have shown that soft constraints can also be used to monitor the shape deformation (figure 1.3). Since many configurations are based on a set of trimmed patches, the corresponding deformation problem is globally under-constrained, and the number of control points is generally far greater than the number of constraints. Usually, objective function indicates a user's design tendency of a curve/surface behavior after deformation. For example, shape fairness is often used to obtain the smoothest and the most graceful shapes, and the criterion corresponds to the minimization of an energy having a physical meaning and leading to natural surfaces. Therefore, soft constraints can be used as a criterion to choose one solution among all those satisfying strict constraints. Hence, soft constraints can be used to help the designer adjust the shape in accordance to complementary parameters that cannot be incorporated into geometric constraints.

This type of constraint, which act on the function $G(X)$ to be minimized, cannot be directly handled by the proposed approach which focuses on the identification of redundancies and conflicts in a set of strict constraints. Those aspects are discussed in the results section.

1.5 From constraints to equations

1.5.1 Expressing constraints with equations

CAD modelers provide their solvers of geometric constraints and usually the solver has its own constraints editor. Basically, the constraints concern vertices of interest, straight lines, planes, circles, spheres, cylinders or freeform curves and surfaces whose parameters are the unknown variables. Constraints ranging from level 1 to level 3 (Section 1.4.1) can be rep-

resented with equations. Those equations can be linear or non-linear. Classical solvers use these constraints to sketch and constrain the shape of desired models. For example, the 2D distance constraint d between two points (x, y) and (x_0, y_0) is translated to the equation $(x-x_0)^2+(y-y_0)^2-d^2=0$. Continuity constraints between two patches can also be represented with equations. Moreover, those mathematical equations can also be represented using computational graph, which is based on Directed Acyclic Graphs (DAGs). In such a representation, a DAG is a tree with shared vertices. The leaves of the tree are either variables (i.e. parameters or unknowns) or numerical coefficients. The internal nodes of the tree are either elementary arithmetic operations or functions such as $exp; sin; cos; tan$. The DAG is also called white box DAG, since it allows for computing the derivatives and hessians automatically. If the mathematical equations associated to geometric constraints are available, it is possible to compute the expressions of the derivatives with formal calculus, which can be resorted to using Grobner basis or Wu-Ritt method if all the constraints are algebraic and can be triangulated into the form $f1(U; x1) = f2(U; x1; x2) =:::= 0$ (U is the parameters vector and x_i are the unknown variables).

1.5.2 Black box constraints

On the contrary, a DAG is called a black box DAG, and a constraint is called a black box constraint when the corresponding constraints cannot be represented with equations or are not computable in practice (Gouaty et al., 2016). This corresponds to constraints of level 4 discussed in section 1.4.1. Examples such as, the maximum of the Von Mises stress should be smaller than 100MPa, the final product should cost less than 100, the mass of the object should be smaller than 100 kilograms, there should not be collisions between the parts, are requirements which cannot be transformed into a set of equations. In the work of (Gouaty et al., 2016), they proposed to use black box DAGs for Variational Geometric Modeling of free-form surfaces and subdivision surfaces and they presented a prototype, DECO, to show the feasibility and promises of this approach. Black box constraints can happen when free-form surfaces are generated tediously from fairly sophisticated modeling functions (e.g. sweep, loft, blend). Of course, these black box constraints cannot be manipulated in the same way as if some equations were available and solvers have to take into account these constraints expressed by functions i.e. constraints requiring the call to a function. In the context of this PhD thesis, we will only consider configurations involving constraints defined by linear and/or non-linear equations. Configurations involving black box constraints have not been addressed.

1.6 Needs for better understanding the design intent

1.6.1 Management of the uncertainties

Today's modelers and solvers cannot fully handle the uncertainties when designers define their requirements. When designing free-form objects, it is impossible to precisely specify a shape at the beginning. Because the idea designers have in mind are usually not fully refined at the conceptual phase but evolves with engineering conditions and simulation results. As a consequence, the PDP requires many back and forth attempts before getting desirable results. Moreover, the results can sometimes be acceptable if they are close to target ones. This can happen when deforming free-form objects, since it inherits problems from the non-linear optimization domain, such as results coverages to local minima rather than global minimum. The distance between local minima and global minimum is uncertain but under the tolerance that designers can accept. Finally, the uncertainties can happen directly on constraints. That is, constraints with inequalities and not only strict constraints. Values of these constraints remain uncertain before solving but are acceptable if their real values after solving are within the range of inequalities.

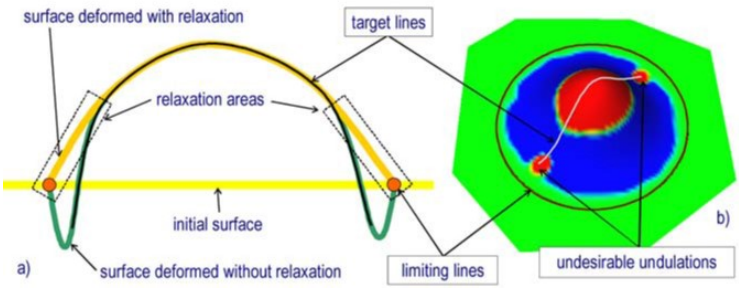


Figure 1.7: Need of relaxation areas (a) to avoid undesirable undulations (b) (Pernot, Qiao, and Veron, 2007).

Pernot et al (Pernot, Qiao, and Veron, 2007) showed an example where the deformation of a surface is constrained by constraint lines. As it is shown in figure 1.7(b), a target line and a limiting line are specified on an initial patch. However, during the drawing of the target line, designer does not have accurate criterion to sketch the end points of the target line. After the deformation, the Gaussian map of the curvature shows the presence of hollows around the end points of the target line. These extremities of the

target line are considered as uncertainty areas which must not be taken into account during the deformation process. Such results are not acceptable for designers. They proposed to relax user-specified constraints with two types of scenarios and explains the effects of relaxation on an amplified configuration in figure 1.7(a). The approach developed in this PhD allows for the detection of redundant and conflicting constraints. Thus, once identified, those configurations can either be deleted or the associated constraints can be relaxed.

1.6.2 Detection and treatment of over-constraints

User-specified requirements may not always be consistent and the overall set can be over-constrained. It is up to the solver to detect those inconsistencies and to give feedbacks on how to remove them.

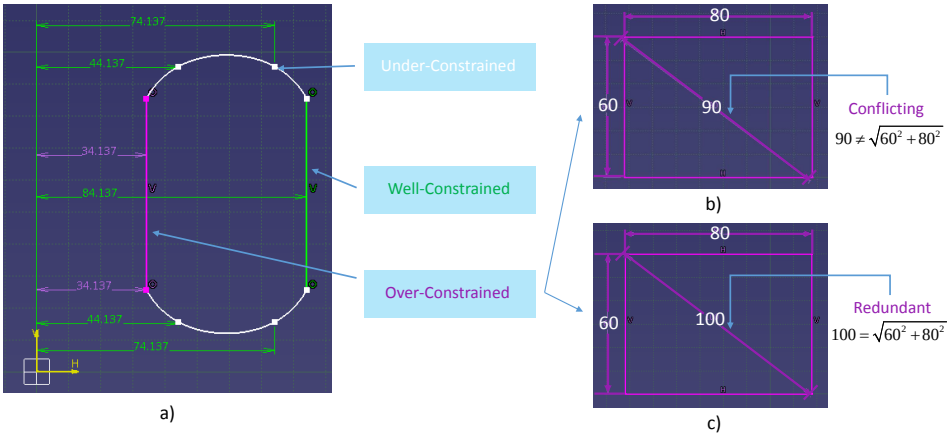


Figure 1.8: 2D sketches a) constraints labeling with colors b) sketch with conflicting constraints c) sketch with redundant constraints. Clearly, the sketcher does not highlight differently redundant/conflicting constraints.

In most of today's modeler, and as is shown in figure 1.8, a geometric configuration can be of three types:

- Under-constrained: number of unknowns is greater than the number of equations. This case happens quite often since designers often insert extra DOFs to satisfy requirements.

- Well-constrained: number of unknowns is equal to the number of equations.

- Over-constrained: number of unknowns is less than the number of equations. And the type of extra equations have two possibilities:
  - redundant: these equations are consistent with the other ones. That is, they do not affect the solution of original system.
  - conflicting: fully inconsistent with the others when constraints express contradictory requirements and lead to no satisfactory solution.

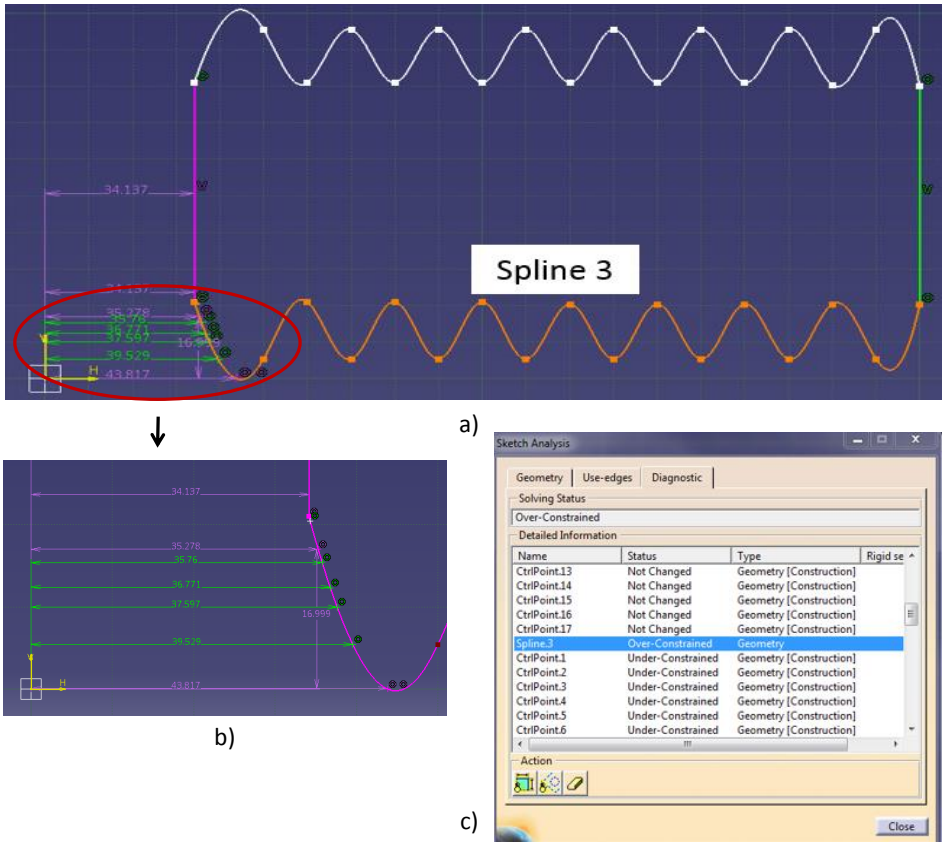


Figure 1.9: A 2D curve sketch a) configuration with curve geometry b) locally over-constrained (according to the local support property) c) globally over-constrained (according to the sketcher)

More specifically, in terms of free-form geometry, the equations do not involve all the variables due to its local support property. Designers may express involuntarily several times the same requirements using different constraints thus leading to redundant equations. But the designers may also involuntarily generates conflicting equations and may have to face over-constrained and unsatisfiable configurations. The configuration of a set of

constraints can however be even more complex: a problem can be globally under-constrained and locally over-constrained (figure 1.9). Tools to detect globally over-constrained configurations exist but are limited to a set of geometric constraints applied on Euler geometries like points, lines, planes, etc (Guillet, 1999). As it is shown in figure 1.8, advanced 2D sketchers available in most commercial CAD software can interactively identify the over-constraints during the sketching process. However, they do not allow for the detection of redundant and/or conflicting constraints. Moreover, the detection of locally over-constrained configurations is much difficult to handle, especially in the case of hybrid geometries composed of polylines, curves, meshes and surfaces. It is not yet possible to analyze the status of 3D NURBS-based equation systems before submitting them to a solver. For example, Spline3 in figure 1.9 is locally over-constrained but globally under-constrained. Constraint analysis of the 2D sketcher shows the whole geometry is globally over-constrained, which is not correct.

Sometimes, over-constrained configurations can be solved by inserting extra degrees of freedom (DoFs) with the Boehm"s knot insertion algorithm. As a consequence, many control points are added in areas where not so many DoFs are necessary (Pernot et al., 2005). This uncontrolled increase of the DoFs impacts the overall quality of the final surfaces which become more difficult to manipulate than the initial ones. Furthermore, some structural over-constraints cannot disappear following this strategy and dedicated decision-support approaches have to be developed to identify and fully manage over-constrained configurations.

As a consequence, this PhD thesis has tried to overcome those limitations while enabling for a proper identification of locally redundant or conflicting configurations.

1.7 Conclusion

In this chapter, we first introduced the structure of PDP and its relationship with the modeling of free-form shapes. Then, we explained modeling multiple requirements in terms of controlling free-form shapes by transforming requirements into constraints and transforming constraints into equations. Since user specified requirements sometimes can be redundant and conflicting, we showed what current modelers can/cannot do with respect to system of constraints and discussed the necessity of debugging the constraints system of free-form configurations. In next chapter, concepts of geometric over-constraints, methods of debugging constraints systems, and the selection of methods for free-form configurations will be discussed.

Chapter 2

Geometric over-constraints detection

In this chapter, background with respect to the modeling of geometric constraints systems are first introduced (section 2.1). Then definitions of geometric over-constraints are summarized and compared in section 2.2. Then, detection methods are introduced and compared in section 2.4, where the evaluation criteria are defined and extracted from geometric constraints solving domain (section 2.3). Some of the methods are tested on different use cases in order to find ones that might be interesting for detecting over-constraints of free form configurations (section 2.5).

.

2.1 Representation of geometric constraints systems

A geometric model can be manipulated either through variables or with geometric entities. This requires a geometric model to be represented either in the equation or geometry levels. In this section, we will show how the models are presented in terms of the two levels.

2.1.1 Graph fundamentals

Graphs are mathematical concepts that have found many uses in computer science. A graph is used to describe a structure where some pairs of objects are involved with some kind of relationships. In constraint systems, objects are geometric entities and relationships are geometric constraints. Therefore, they are commonly used in geometric constraints solving

domain. For example, bipartite graph is used in the work of (Ait-Aoudia, Jegou, and Michelucci, 2014) whereas (Fudos and Hoffmann, 1997) use directed graph. These two types of graphs are mostly used in literature. However, we also include directed graph in this section because it is often used in constraint solving process and will be used in the later chapters of this manuscript as well.

Bipartite graph

Definition In graph theory, a bipartite graph ($G = (U \cup V, E)$) is a graph whose vertices can be divided into two disjoint sets U and V (that is, U and V are independent sets) such that every edge e ($e \in E$) connects u ($u \in U$) to v ($v \in V$). Vertex sets U and V are usually called the parts of the graph. Equivalently, a bipartite graph is a graph that does not contain any odd-length cycles (Diestel, 2006). For example, figure 2.1-a is a bipartite graph, where U is a set of applicants and V is a set of jobs. In terms of geometric constraints system, U and V can either be equations and variables, or constraints and entities depending on the level of modeling.

Matching problems are often concerned with bipartite graphs. A matching in a bipartite graph is a set of the edges chosen in such a way that no two edges share an endpoint. For a geometric constraints system, a matching happens between one variable and one equation or between one constraint and one entity. The unmatched ones within such bipartite graph enables to know if there are constraints/equations or entities/variables left, which indicates the constrained status of a system. To find them, we need to take advantage of Maximum bipartite matching at the level of equations or Maximum weighted bipartite matching at the level of geometries.

Maximum bipartite matching A maximum matching is a matching of maximum size (maximum number of edges). In a maximum matching, if any edge is added to it, it is no longer a matching. There can be more than one maximum matching for a given bipartite graph. For example, figure 2.1-b shows a maximum matching found for the bipartite graph in figure 2.1-a, meaning that there are maximum five people who can get jobs. In this example, an applicant can get one job, and reversely a job can only be assigned to one person. The maximum matching can be solved by converting a bipartite graph into a flow network (figure 2.2-a) and then using Ford-Fulkerson algorithm (Ford Jr and Fulkerson, 2009) to find the maximum flow in the flow network.

The method is used to analyze the structure of equation systems. The

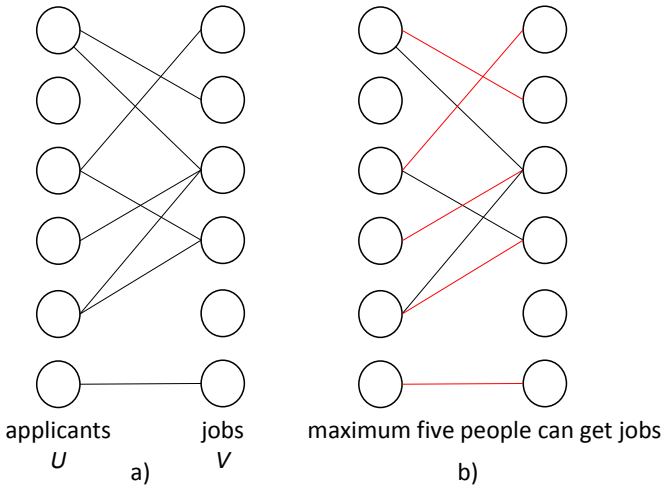


Figure 2.1: a) bipartite graph b) bipartite graph with maximum matching(red lines)

maximum matching approach was first used by Serrano (Serrano, 1991) for systems of non-linear equations appearing in conceptual design problems. Also, it has been adopted by Dulmage-Mendelsohn decomposition algorithm to debug a geometric constraints system (Dulmage and Mendelsohn, 1958). However, if a system is represented not at the level of equations but at the level of geometries, then the method is not applicable. In this case, Maximum weighted bipartite matching is used, which will be discussed in next paragraph.

Maximum weighted bipartite matching Maximum weighted bipartite matching is a generalization of maximum bipartite matching. For a bipartite graph $G = (U \cup V, E)$, and edge weights $w_{i,j}$, find a matching of maximum total weight. These weights are used as variables during the process of maximizing the total weight. The optimization process can be solved by algorithms like Negative cycles, hungarian method, and Prime dual method (Bang-Jensen and Gutin, 2008). Finding the maximum weighted bipartite matching has been used by Latham et al. to analyze the connectivity of a constraints system (Latham and Middleditch, 1996), which is modeled directly at the level of geometries.

Directed graph

A graph can be directed. For example, if the vertices represent people at a party, and there is a directed edge from a person A to a person B corresponds

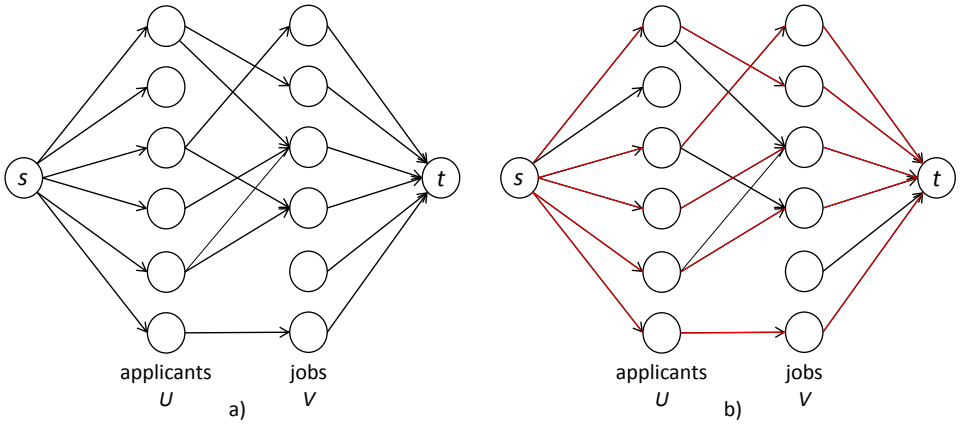


Figure 2.2: a) building a flow network b) finding a maximum matching by computing the maximum flow from source to sink

to A's admiring B, because admiration is not necessarily reciprocated. In graph theory, a directed graph $G = (V, E)$ is a graph that is a set of vertices V connected by edges E, where the edges have a direction associated with them. More specifically, a finite directed graph with loops is addressed as a loop-digraph, while one without loops is addressed as a directed acyclic graph (DAG). A DAG is a directed graph that starting at any vertex v and following a consistently-directed sequence of edges cannot loop back to v again (Thulasiraman and Swamy, 2011). Graphs of figure 2.3 are all directed graphs. More precisely, figure 2.3-a is a DAG but figure 2.3-b is not, since the latter has a loop inside the structure. Directed graphs are often used as a flow network for computing the maximum flow or locating subgraphs satisfying certain conditions (figure 2.2).

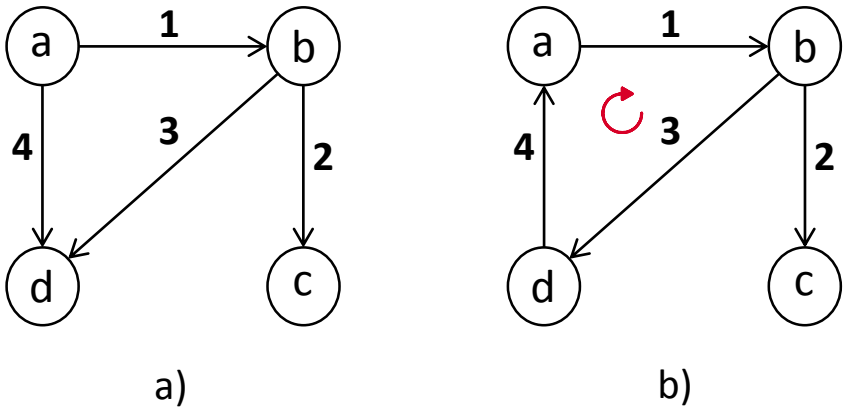


Figure 2.3: a) A DAG b) A loop-digraph

Strongly connected component A strongly connected component is a maximal subgraph of a directed graph G such that for every pair of vertices (u, v) in the subgraph, there is a directed path from u to v and a directed path from v to u. A strongly connected component of G is an induced subgraph which is strongly connected and no additional edges or vertices from G can be included in the subgraph without breaking its property of being strongly connected (figure 2.4-b). Strongly connected components are also used to compute the Dulmage–Mendelsohn decomposition, a classification of the edges of a bipartite graph, according to whether or not they can be part of a perfect matching in the graph (Dulmage and Mendelsohn, 1958). A perfect matching of a graph is a matching (i.e., an independent edge set) in which every vertex of the graph is incident to exactly one edge of the matching. Therefore, it is a matching containing n/2 edges (the largest possible), meaning perfect matchings are only possible on graphs with an even number of vertices. Strongly connected components can be found by Breadth First Search (BFS) or Depth First Search (DFS) in linear time (Tarjan, 1972).

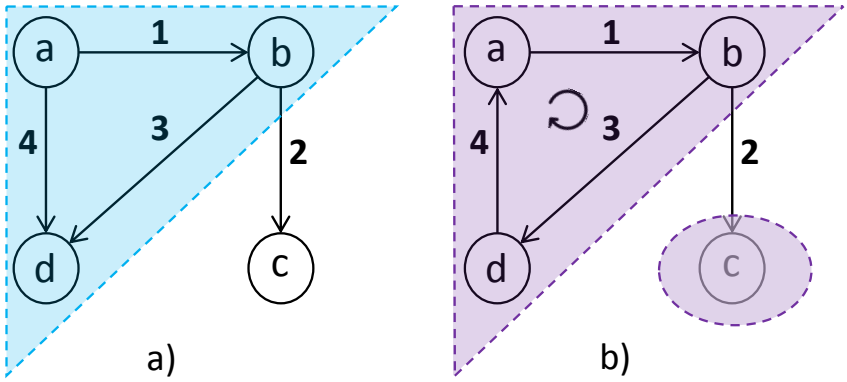


Figure 2.4: a) Weakly connected component marked blue b) Strongly connected components marked purple

Weakly connected component A weakly connected component is a maximal subgraph of a directed graph G such that for every pair of vertices (u, v) in the subgraph, there is an undirected path from u to v and a directed path from v to u. A weakly connected component of G is an induced subgraph which is weakly connected and no additional edges or vertices from G can be included in the subgraph without breaking its property of being weakly connected. The subgraph marked blue of figure 2.4-a is a weakly connected component. As far as we know, weakly connected components are not used by any method of geometric constraint solving. But in this manuscript, we propose an algorithm to find a set of constraints to which

an over-constraint is redundant or conflicting (section 3.4).

Undirected graph

A graph can also be undirected. For example, if the vertices represent people at a party, and there is an edge between two people if they shake hands, then this graph is undirected because any person A can shake hands with a person B only if B also shakes hands with A. An undirected graph is a graph in which edges have no orientation. The edge from v to u is identical to the edge from u to v, i.e., they are not ordered pairs (figure 2.5). In geometric constraints solving, it is often used as constraint graph to initially represent a constraints system (section 2.1.3).

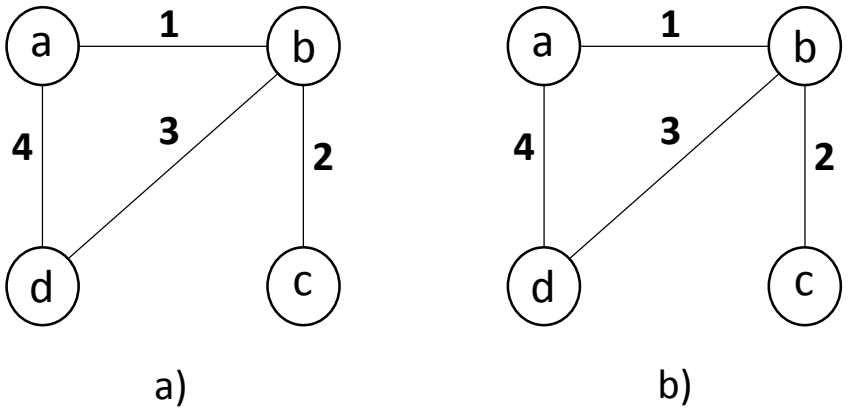


Figure 2.5: a) Undirected graph of figure 2.4-a b) Undirected graph of figure 2.4-b

Matrix

Apart from graph representation, matrix is another format for showing the relationships between related objects in graph theory. Depending on the type of objects, matrix can either be incidence matrix or adjacency matrix.

Incidence matrix Incidence matrix shows the relationship between edges and vertices of a graph. For a directed graph $G = (V, E)$ with $V = \{v_1, \cdots, v_n\}$ and $E = \{e_1, \cdots, e_m\}$, the incidence matrix is a $n \times m$ matrix B, such that $B_{i,j} = 1$ if the edge e_j leaves vertex v_i, -1 if it enters vertex v_i, and 0 otherwise (figure 2.6). For an undirected graph $G = (V, E)$ with $V = \{v_1, \cdots, v_n\}$ and $E = \{e_1, \cdots, e_m\}$, the incidence matrix is a $n \times m$ matrix B, such that $B_{i,j} = 1$ if the edge e_j connects vertex v_i and 0 otherwise (figure 2.7).

Incidence matrix is mainly used for practical computation of an algorithm since it stores information between vertices and edges of a structure. Sometimes, it is also interesting to know the relationship between vertices of a structure before design a sophisticated algorithm. For example, if most of the vertices of a structure are unconnected, then the structure can be decomposed into small parts, and each part can be further analyzed. In such case, adjacent matrix for representing a system is applied.

	a	b	c	d
1	1	-1	0	0
2	0	1	-1	0
3	0	1	0	-1
4	1	0	0	-1

a)

	a	b	c	d
1	1	-1	0	0
2	0	1	-1	0
3	0	1	0	-1
4	-1	0	0	1

b)

Figure 2.6: a) Incidence matrix of figure 2.4-a b) Incidence matrix of figure 2.4-b

	a	b	c	d
1	1	1	0	0
2	0	1	1	0
3	0	1	0	1
4	1	0	0	1

a)

	a	b	c	d
1	1	1	0	0
2	0	1	1	0
3	0	1	0	1
4	1	0	0	1

b)

Figure 2.7: a) Incidence matrix of figure 2.5-a b) Incidence matrix of figure 2.5-b

Adjacency matrix Adjacency matrix represents relationship between pair of vertices of a graph. As a result, it is a square matrix. For an undirected graph without self-loops, the adjacency matrix is a square $|V| \times |V|$ matrix A such that its element A_{ij} is 1 when there is an edge between vertex V_i and vertex V_j, and 0 when there is no edge. For a directed graph without

self-loops, the adjacency matrix is a square $|V| \times |V|$ matrix A such that its element A_{ij} is 1 when there is an edge from vertex V_i to vertex V_j, -1 when the edge is directed from vertex V_j to vertex V_i, and 0 when there is no edge (figure 2.8). Here, $|V|$ represents the number of elements in V. For an undirected graph, the adjacency matrix is a square $|V| \times |V|$ matrix A such that its element A_{ij} is 1 when there is an edge between vertex V_i and vertex V_j, and 0 when there is no edge (figure 2.9).

	a	b	c	d
a	0	1	0	1
b	-1	0	1	1
c	0	-1	0	0
d	-1	-1	0	0

a)

	a	b	c	d
a	0	1	0	-1
b	-1	0	1	1
c	0	-1	0	0
d	1	-1	0	0

b)

Figure 2.8: a) Adjacency matrix of figure 2.4-a b) Adjacency matrix of figure 2.4-b

	a	b	c	d
a	0	1	0	1
b	1	0	1	1
c	0	1	0	0
d	1	1	0	0

a)

	a	b	c	d
a	0	1	0	1
b	1	0	1	1
c	0	1	0	0
d	1	1	0	0

b)

Figure 2.9: a) Adjacency matrix of figure 2.5-a b) Adjacency matrix of figure 2.5-b

All adjacency and incidence matrices are totally unimodular. It permits Linear Programming for computing flows or maximum matching to work.

2.1.2 Graphs at the level of equations

In this section, we give an example of a geometric constraints system by modeling it at the level of equations. As discussed in section 1.5.1, the method use variables to describe shape, position etc. of geometries as well as algebraic equations to show their relationships. Consequently, a constraint system is represented by a group of algebraic equations.

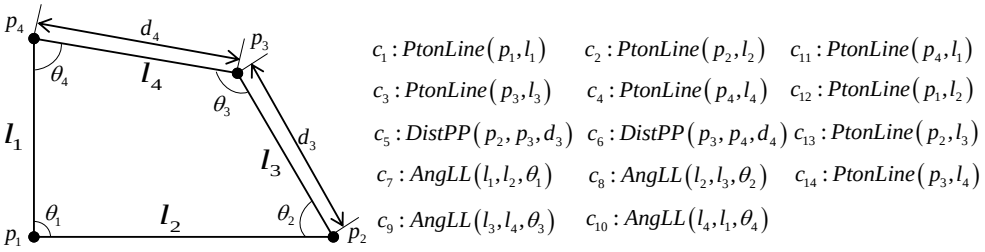


Figure 2.10: A simple 2D sketch

For example, figure 2.10 is a 2D sketch with 14 constraints. If geometries are represented with variables: $p_i, (x_i, y_i)_{i=1\cdots4}$; $l_i, (y = a_i \times x + b_i)_{i=1\cdots4}$, then algebraic equations are:

$$
\begin{aligned}
&e_1 : a_1 \cdot x_1 + b_1 = y_1; e_2 : a_2 \cdot x_2 + b_2 = y_2 \\
&e_3 : a_3 \cdot x_3 + b_3 = y_3; e_4 : a_4 \cdot x_4 + b_4 = y_4 \\
&e_5 : (x_3 - x_2)^2 + (y_3 - y_2)^2 = d_3^2 \\
&e_6 : (x_4 - x_3)^2 + (y_4 - y_3)^2 = d_4^2 \\
&e_7 : cos(\theta_1) \cdot \| \overrightarrow{p_2 - p_1} \| \cdot \| \overrightarrow{p_4 - p_1} \| = \overrightarrow{p_2 - p_1} \cdot \overrightarrow{p_4 - p_1} \\
&e_8 : cos(\theta_2) \cdot \| \overrightarrow{p_1 - p_2} \| \cdot \| \overrightarrow{p_3 - p_2} \| = \overrightarrow{p_1 - p_2} \cdot \overrightarrow{p_3 - p_2} \\
&e_9 : cos(\theta_3) \cdot \| \overrightarrow{p_2 - p_3} \| \cdot \| \overrightarrow{p_4 - p_3} \| = \overrightarrow{p_2 - p_3} \cdot \overrightarrow{p_4 - p_3} \\
&e_{10} : cos(\theta_4) \cdot \| \overrightarrow{p_1 - p_4} \| \cdot \| \overrightarrow{p_3 - p_4} \| = \overrightarrow{p_1 - p_4} \cdot \overrightarrow{p_3 - p_4} \\
&e_{11} : a_1 \cdot x_4 + b_1 = y_4; e_{12} : a_2 \cdot x_1 + b_2 = y_1 \\
&e_{13} : a_3 \cdot x_2 + b_3 = y_2; e_{14} : a_4 \cdot x_3 + b_4 = y_3
\end{aligned}
\tag{2.1}
$$

The equations can then be analyzed by using numerical methods. However, they can further be transformed into an equation graph, if structural analysis methods are to be used.

An equation graph is a bipartite graph where two classes of nodes represent equations and variables respectively. In this case, the equation graph of equations 2.1 is shown in figure 2.11.

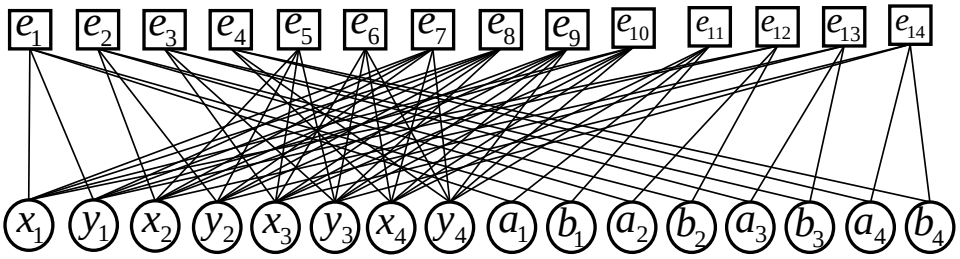


Figure 2.11: Bipartite graph of equations 2.1

2.1.3 Graphs at the level of geometries

The above system can also be represented at the level of geometries. Extract geometric entities and constraints into vertices of a graph provides another way of system modeling.

Bipartite graph Similar to figure 2.11, bipartite graph can model a constraint system by using two classes of vertices to represent geometric entities and constraints respectively as well as edges to show their relationships (figure 2.12).

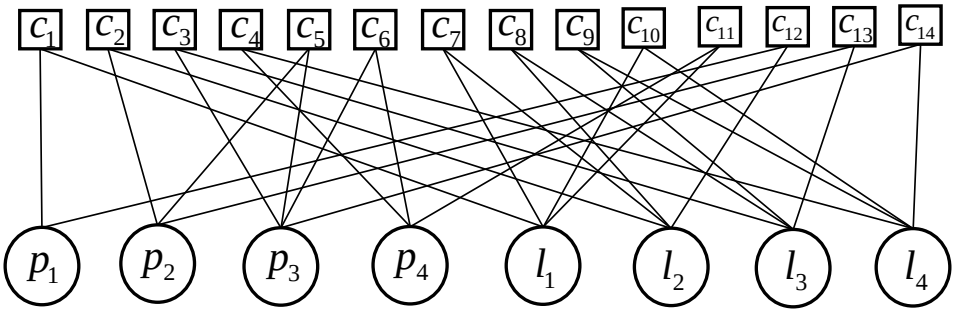


Figure 2.12: Bipartite graph of configuration in figure 2.10

Constraint graph Bipartite graph is visually cumbersome and not intuitive. Some people prefer the constraint graph with vertices representing geometric entity and edges representing constraints that visually more 'natural' in the level of geometry. Constraint graph use vertices to represent geometric entities only and edges to represent constraints between. The constraint graph of figure 2.10 is shown in figure 2.13.

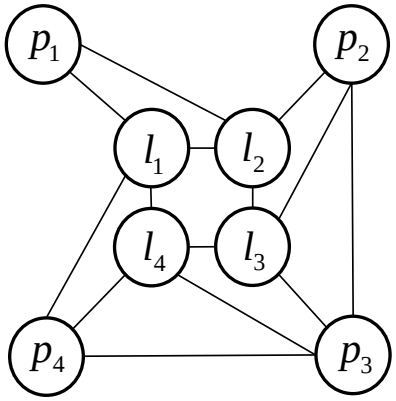


Figure 2.13: Constraint graph of configuration in figure 2.10

From above examples, we can see that modeling geometric systems at the level of geometries is more intuitive than at the level of equations. But it ignores numerical information of a system, since the latter generates concrete equations. The selection of the two ways depends on users' practical needs.

2.2 Basic definitions

This section introduces several key concepts in the manuscript. The first one is geometric over-constraints which are defined both at the level of geometries and equations. Also, it is necessary to introduce singular configurations since they often confuse the judgment of geometric over-constrained configurations. After that, definition of over-constraints in terms of free form geometry are formalized.

2.2.1 Geometric over-constraints at the level of geometries

At this level, definitions are classified into two groups: constraint graph group and bipartite graph group. For the former, the constraint graph is transformed into a weighted constraint graph, where the weight of a vertex represents DoFs of an entity and the weight of an edge represents DoFs removed by a constraint. For the bipartite graph group, only the weight of vertices are added: the weight of an entity equals to its DoFs and the weight of a constraint equals to the DoFs it can remove.

Definitions with respect to constraint graph

Here, we use $G = (V, E)$ to represent a constraint system with $\mid V \mid$ number of entities and $\mid E \mid$ number of constraints.

In Rigidity Theory (*Combinatorial Rigidity*), Laman's theorem (Laman, 1970) characterizes the rigidity of bar frameworks, where a geometric system is composed of points constrained by distances.

Theorem 1 A constraint system in the 2D plane composed of N points linked by M distances is rigid iff $2 \cdot N - M = 3$ and for any subsystem composed of n points and m distances, $2 \cdot n - m \geq 3$.

The constraints and entities are limited to distances and points respectively. Podgorelec (Podgorelec, Žalik, and Domiter, 2008) extended the theorem by assuming that each geometric element has 2 DoFs and each constraint eliminates 1 DoF. Therefore, the weight of vertices and edges are of the constraint graph is 2 and 1 respectively.

Definition 1 For constraint graph $G = (V, E)$, a geometric constraint system is:

- *Structurally over-constrained* if there is a subgraph $G' = (V', E')$ with $1 \cdot |E'| > 2 \cdot |V'| - 3$,
- *Structurally under-constrained* if G is not *structurally over-constrained* and $1 \cdot |E| < 2 \cdot |V| - 3$, or
- *Structurally well-constrained* if G is not *structurally over-constrained* and $1 \cdot |E| = 2 \cdot |V| - 3$.

Definition 2 A constraint e is a *structural over-constraint* if a structurally over-constrained subsystem $G' = (V', E')$ of G with $e \in E'$, can be derived such that $G'' = (V', E' - e)$ is structurally well-constrained.

An example is given to illustrate **Definition 1**. The system is composed of 3 points (each has 2 dofs) with different constraints in 2D space. For figure 2.14-a, it is over-constrained because it contains 3 distance constraints and 3 vertical position constrains. Since each consumes 1 dof, the total system consumes 6 dofs, satisfying $6 > 2 \times 3 - 3$. Figure 2.14-b is structurally well-constrained since the constraints are reduced into 3 distance constraints, which satisfying $3 = 2 \times 3 - 3$. Figure 2.14-c is structurally under-constrained since only 2 distance constraints are left, satisfying $3 < 2 \times 3 - 2$.

Definition 1 is correct if only all geometric entities are points and all constraints are distances in 2D. It cannot be used to characterize geometric constraint systems where constraints other than distance constraints are involved. For example, in the case of angle constraints in 2D: 3 line segments with 3 incidence constraints form a triangle with $3 \cdot 4 - 3 \cdot 2 = 6$ DoFs. If added 3 angle constraints(each remove 1 dof), the system will be *Structurally well-constrained* according to **Definition 1**. But in fact, 2 angle constraints is enough since the third one is linear combination of the other two. Another counter example is the double banana geometry, where segments represent point-point distances in 3D, which will be discussed in section 2.5.2.

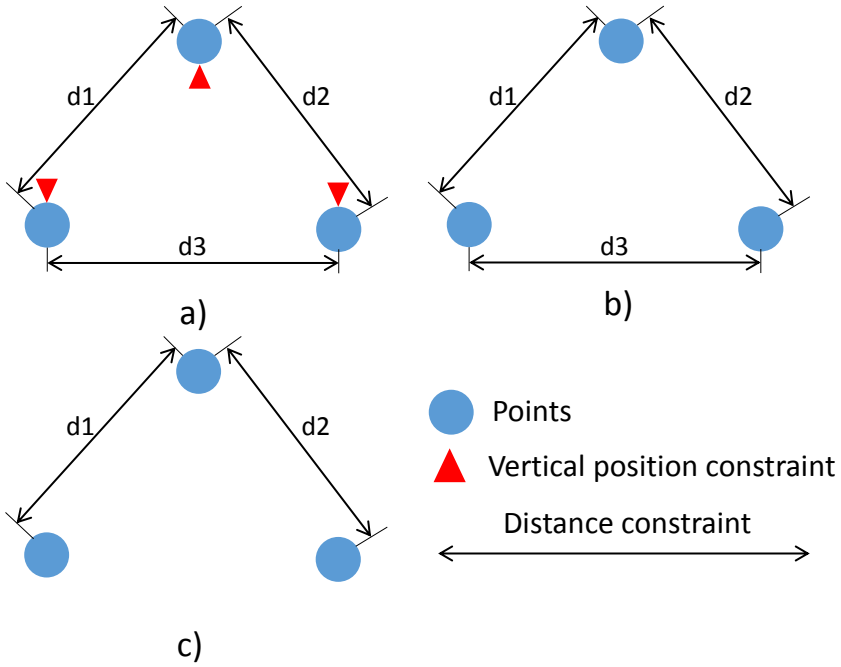


Figure 2.14: a) Structurally over-constrained b) Structurally well-constrained c) Structurally under-constrained

To extend the definition to 3 dimensions and deal with double banana geometry properly, Sitharam and Zhou (Sitharam and Zhou, 2004) introduced a set of new definitions. First, they replaced value 3 in the **Definition 1** with D, which is a function of dimension d: $D = (d+1) * d/2$.

Definition 3 Degree of freedom (DoF) of a geometry entity ($DoF(v)$, v is the geometry) is the number of independent parameters that must be set to determine its position and orientation. For a system $G = (V, E)$, its DoFs is defined as $DoF(G) = \sum_{v \in V} DoF(v)$.

Definition 4 Degree of freedom of a geometric constraint ($DoC(e)$, e is the constraint) is the number of independent equations needed to represent it. For a system $G = (V, E)$, the DoFs all constraints can remove is $DoC(E) = \sum_{e \in E} DoC(e)$.

Definition 5 For constraint graph $G = (V, E)$, a geometric constraint system is:

- *Structurally over-constrained* if there is a subgraph $G' = (V', E')$ satisfying $DoC(E') > DoF(V') - D$,
- *Structurally well-constrained* if $DoC(E) = DoF(V) - D$ and all subgraphs $G' = (V', E')$ satisfying $DoC(E') \leq DoF(V') - D$, or
- *Structurally under-constrained* if $DoC(E) < DoF(V) - D$ and contains no *structurally over-constrained* subgraphs.

A typical example that **Definition 5** cannot treat properly is 2 points binding with distance constraint in 3D. It allows only 5 of 6 possible independent displacements since the system cannot rotate around axis crossing the 2 points. More counter examples in (Jermann, Neveu, and Trombettoni, 2003) suggest that the value of D depends on the system itself rather than dimension. Therefore, Jerman et al introduced *Degree of Rigidity* (DoR) to replace the DoFs a system is expected to have if it is rigid. Their definitions are as follows.

Definition 6 For constraint graph $G = (V, E)$, a geometric constraint system is:

- *Structurally over-constrained* if there is a subgraph $G' = (V', E')$ satisfying $DoC(E') > DoF(V') - DoR(V')$,
- *Structurally well-constrained* if $DoC(E) = DoF(V) - DoR(V)$ and all subgraphs $G' = (V', E')$ satisfying $DoC(E') \leq DoF(V') - DoR(V')$, or
- *Structurally under-constrained* if $DoC(E) < DoF(V) - DoR(V)$ and contains no *structurally over-constrained* subgraphs.

The rule of computing DoR is described in (Jermann, Neveu, and Trombettoni, 2003). Within the rule, for two secant planes in 3D, the DoR is 5 while for two parallel planes is 4. Similarly, the DoR of 3 collinear points is 2, while the DoR of 3 non collinear points is 3.

A pure graphbased method has no mean to know if 3 points are collinear or not, or if two planes are parallel or not. It either assumes the configuration is generic or it can try to look if the parallelism/collinearity is an explicit constraint of the system; but it may happen that the parallelism/collinearity is a remote consequence of a set of constraints, thanks to Desargues, or Pappus, or Pascal, or Miquel theorems: the incidence in the conclusion is a non trivial consequence of the hypothesis. This will be further discussed in section 2.2.1.

Definitions with respect to bipartite graph

Latham et al (Latham and Middleditch, 1996) introduced similar definitions based on a connectivity graph. It is a graph where vertices represent geometric entities and constraints, which can be treated as a bipartite graph. Note that, we use $G = (U, V, E)$ to denote a bipartite graph whose partition has the vertices U(entities) and V(constraints), with E denoting the edges of the graph.

Definition 7 For bipartite graph $G = (U, V, E)$, a geometric constraint system is:

- *Structurally over-constrained* if it contains an unsaturated constraint,
- *Structurally under-constrained* if it contains an unsaturated entity.

A vertex u or v is said to be unsaturated if $DoF(u)$ or $DoC(v)$ is not equal to the number of weights of incident edges in a maximal weighting. And an unsaturated constraint is treated as a *structural over-constraint*. The weights of edges are computed by maximal weighted matching of a bipartite graph. In section 2.4.1, an example will be given to illustrate the definition.

Geometric redundancy

Geometric redundancy refers to those additional constraints trying to constrain internally established relations. The relations are consequences of domain-dependent mathematical theorems hidden in a geometric configuration. Users are typically not aware of these implicit constraints and will always try to constrain the internal established relations by additional constraints.

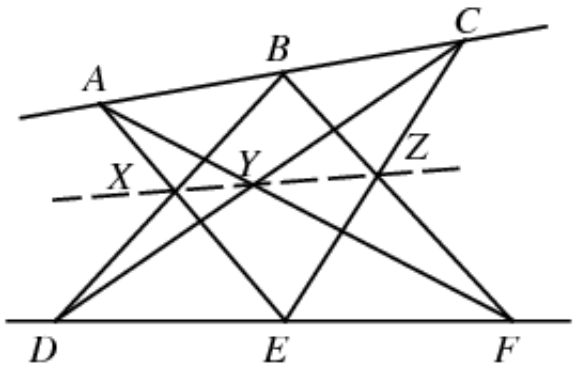


Figure 2.15: Pappus's hexagon theorem: Points X, Y and Z are collinear on the Pappus line (dotted line). The hexagon is $AFBDCE$.

Geometric redundancy does not use parameters. Therefore, this type of geometric over-constraints cannot be detected by methods based on Dof-counting. In 2D, a typical example is the 3-angles constraints specified on a triangle. Obviously, total value of three angles equals to 180°. It is not necessary to specify all three angles as constraints because the value of third one can be easily derived once the values of other two angles are defined. Therefore, specifying the 3-angles constraints will generate a geometric redundancy that is either redundant or conflicting. In 3D, every incidence theorem(Desargues, Pappus, Pascal etc) provides implicit dependent constraints (Michelucci and Foufou, 2006a). For example, Pappus's hexagon theorem (Coxeter and Greitzer, 1967) states that given one set of collinear points A, B, C, and another set of collinear points D, E, F, then the intersection points X, Y, Z of line pairs AE and DB, AF and DC, BF and EC are collinear, lying on the Pappus line. In this case, if specifying line pairs XY and YZ to be collinear, then this constraint is the geometric redundancy (figure 2.15).

Geometric over-constraints at the level of equations

In this section, we summarize the definitions used when a qualitative study of geometric systems is to be performed at the level of equation. Modeling in geometric level is geometric entity oriented, which preserves geometric information of the system. Modeling at the level of equations, however, discards geometric properties of a system but enables a fine detection of geometric over-constraints.

Structural definition System of equations are transformed into bipartite graph, where vertices represent equations and variables respectively (figure 2.11). The characterization is based on the results of maximum matching (Ait-Aoudia, Jegou, and Michelucci, 2014). Here, we assume that $G = (U, V, E)$ is a bipartite graph with U and V ($U \cap V = \varnothing$) representing variables and equations respectively, and E representing edges.

Definition 8 For bipartite graph $G = (U, V, E)$ and its subgraph $G' = (U', V', E')$. G' is:

- *Structurally over-constrained* if the number of elements in U' is smaller (in cardinality) than the number of V'. i.e. $\mid U^{'} \mid < \mid V^{'} \mid$
- *Structurally well-constrained* iff G' has perfect matching.
- *Structurally under-constrained* if the number of elements in U' is larger (in cardinality) than the number of elements in V'. i.e. $\mid U^{'} \mid > \mid V^{'} \mid$

Definition 9 Let M be a maximum matching of $G = (U, V, E)$. If M is not perfect matching and V' is the subset of V which is not saturated by M, then equations of V' are the *Structural over-constraints.*

Numerical definition

Informally, an over-constrained problem has no solutions, a well-constrain ed problem has a finite number of solutions, but an under-constrained problem has infinitively many solutions. Based on the Matroid theory (Oxley, 2006), we give definitions of basis equations, redundant and conflicting equation as follows.

Definition 10 Let $G = (E, V, P)$ be a geometric constraints system, where E is a set of equations, V is a set of variables and P is a set of parameters. Let E_r be a non-empty collection of subsets of E, called **basis equations** (we call it **basis** in short), satisfying:

- no **basis** properly contains another **basis**;
- if E_{r1} and E_{r2} are **basis** respectively and if e is any equation of E_{r1}, then there is an equation f of E_{r2} such that $\{(E_{r1} - e) \cup f\}$ is also a **basis**.

Definition 11 Let $G = (E, V, P)$ be a geometric constraints system. Let E_r be a **basis**. For an equation e, adding it to E_r forming a new group: $\{E_r \cup e\}$. If $\{E_r \cup e\}$ is solvable, then e is a **redundant equation**.

Definition 12 Let $G = (E, V, P)$ be a geometric constraints system. Let E_r be a **basis**. For an equation e, adding it to E_r forming a new group: $\{E_r \cup e\}$. If $\{E_r \cup e\}$ is non-solvable, then e is a **conflicting equation**.

Definition 13 Let $G = (E, V, P)$ be a geometric constraints system which is composed of two subsystems: $G_b = (E_b, V, P)$ and $G_o = (E_o, V, P)$ with $\{E = E_b \cup E_o, E_b \cap E_o = \varnothing\}$. If E_b is a **basis**, then E_o is a set of **numerical over-constraints**.

Property For an over-constraint $E_{oi} \in E_o$, the *Spanning Group* E_{sg} of E_{oi} is a group of independent constraints, with which E_{oi} is redundant or conflicting. For linear systems, the spanning group $E_{sg} = \{e_{sg1}, e_{sg2}, \cdots, e_{sgn}\} \subset E_b$ of E_{oi} satisfies:

$$E_{oi} = \sum_{j=1}^{n} c_j e_{sgj} + b \tag{2.2}$$

where $c_j \neq 0$ and is the corresponding scalar coefficient, $\{e_{sg1}, e_{sg2}, \cdots, e_{sgn}\}$ are linear independent and b is the bias. Thus, E_{oi} is a linear combination of $\{e_{sg1}, e_{sg2}, \cdots, e_{sgn}, b\}$. Moreover, E_{oi} is redundant if $b = 0$ otherwise it is conflicting.

However, E_{sg} is not unique for a given E_{oi}. For example, assuming a linear system of constraints represented at the level of equations:

$$\begin{aligned} e1 &: x_1 + x_2 + x_3 + x_4 = 1 \\ e2 &: x_1 + 2x_2 + 3x_3 + x_4 = 4 \\ e3 &: x_1 - 2x_2 + x_3 + x_4 = 5 \\ e4 &: 6x_1 + x_3 + 2x_4 = 7 \\ e5 &: 8x_1 + 5x_3 + 4x_4 = 17 \\ e6 &: 11x_1 + x_2 + 10x_3 + 7x_4 = 27 \end{aligned} \tag{2.3}$$

Clearly, the system is over-constrained since there are more equations than variables. Through linear analysis of the system, we find that $e5$ is a linear combination of $\{e2, e3, e4, 1\}$ and is spanned by $\{e2, e3, e4\}$; $e6$ is a linear combination of $\{e1, e2, e3, e4, 1\}$ and is spanned by $\{e1, e2, e3, e4\}$ (figure 2.16). Since the bias of the two groups is 1, both $e5$ and $e6$ are conflicting. In this case, $\{e1, e2, e3, e4\}$ can be treated as a set of basis constraints

since all the equations are independent and the number of them equals to the number of variables.

$$\begin{aligned} e1 : x_1 + x_2 + x_3 + x_4 &= 1 \\ e2 : x_1 + 2x_2 + 3x_3 + x_4 &= 4 \\ e3 : x_1 - 2x_2 + x_3 + x_4 &= 5 \\ e4 : 6x_1 + x_3 + 2x_4 &= 7 \end{aligned} \tag{2.4}$$

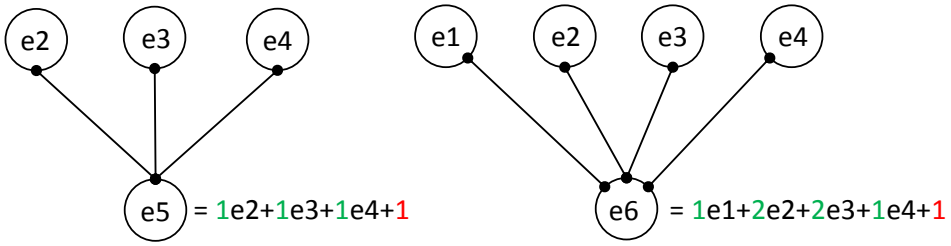


Figure 2.16: Spanning group of $e5$ and $e6$: numbers marked green are coefficients while the ones marked red are the biases

However, if we replace $e4$ with $e5$, the new set $\{e1, e2, e3, e5\}$ is also the basis constraints set satisfying **Definition 13**. Linear analysis result shows that $e4$ is a linear combination of $\{e2, e3, e5, -1\}$ and is spanned by $\{e2, e3, e5\}$; $e6$ is a linear combination of $\{e1, e2, e3, e5\}$ and is spanned by $\{e1, e2, e3, e5\}$ (figure 2.17). Also, $e4$ is conflicting and $e5$ is redundant according to the corresponding bias values.

$$\begin{aligned} e1 : x_1 + x_2 + x_3 + x_4 &= 1 \\ e2 : x_1 + 2x_2 + 3x_3 + x_4 &= 4 \\ e3 : x_1 - 2x_2 + x_3 + x_4 &= 5 \\ e5 : 8x_1 + 5x_3 + 4x_4 &= 17 \end{aligned} \tag{2.5}$$

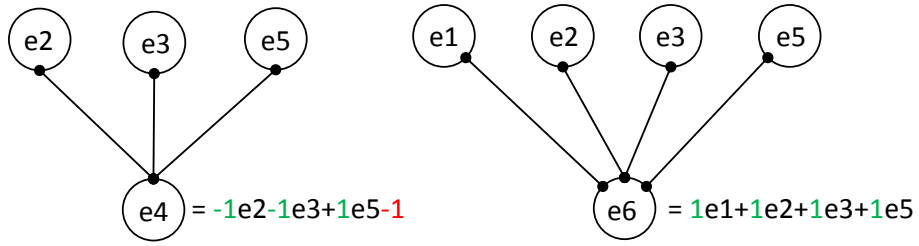


Figure 2.17: Spanning group of $e4$ and $e6$: numbers marked green are coefficients while the one marked red is the bias

From the figure 2.16 and figure 2.17, we can see that the spanning group of $e6$ is not unique, which depends on the set of basis constraints. Also, the type of over-constraint can change: $e6$ is conflicting for basis constraints set 2.4 while redundant for basis constraints set 2.5.

2.2.2 Evaluation of the definitions

A set of criteria are defined to evaluate these definitions (table 2.1). These criteria corresponds to the columns of the table, which are: D is a dimension (system)-dependent constant discussed in section 2.2.1; geometries refer to the geometric type a definition used to specify; counter example lists of geometries that a definition cannot deal with; fixation is used to highlight definitions that includes determining the location and orientation of a geometric configuration; geometric redundancy is to distinguish definitions that cannot be mislead by geometric redundancy.

	D	geometries	constraints	counter example	fixation	geometric redundancy
Def 1	3	points	distances	double banana	no	no
Def 5	0,3,6	points	distances	ex1	no	no
Def 6	DoR	points,lines,planes	distances, incidencies	ex2	no	no
Def 7	0	any	any	?	yes	no
Def 8	0	any	any	black box constraints	yes	no
Def 10	0	any	any	black box constraints	yes	no

ex1: 2 points binding with distance constraint in 3D
ex2: configurations with geometric redundancy

Table 2.1: Evaluations of definitions

From the table, we can see that definitions can de divided into two groups: Def 1,5,6 and Def 7,8,10, either based on criteria D or on fixation. Because the former group manipulate geometric elements directly at the level of geometries and geometric constraints are usually supposed to be independent of all coordinates system, they can not be used to determine the location and orientation of a geometric configuration (no fixation) as well as carefully defined the value of D. Also, the defined type of geometries and constraints are limited. For example, Def 1 and Def 5 are defined for points geometries and distances constraints only. But collinear (and cocyclic, coconic, cocubic, etc.) points are forbidden. Def 6 extends the type of geometries to points, lines and planes as well as it allows for incidence constraints to be defined. Def 7 extracts geometric entities and constraints to DoFs and DoCs, and define over-constraints by simply comparing the number of

*DoF*s of geometric entities and *DoC*s of geometric constraints. In this way, the definition is not limited to any specific class of geometric entities and constraints. Def 8,10, however, are numerical definitions manipulating geometries and constraints at the level of equations. These definitions can cover any geometric entities and constraints once they can be transformed into equations. The counter examples are black box constraints (section 1.5.2) which cannot be represented with equations. Moreover, these definitions expect systems to be fixed with respect to a global coordinate system and thus $D = 0$. Finally, since geometric redundancy does not use parameters of any geometries of a constraints system, it cannot be covered by any of these definitions.

2.3 Criteria for evaluating the approaches

To carry out appropriate analyses and comparisons between the over-constraints detection approaches, various evaluation criteria and a ranking system are here proposed. They depend on the application domain needs and characteristics. Considering the detection process as well as users' needs for debugging, these approaches have been classified into five main categories: criteria related to the process of detecting over-constraints; criteria related to the system decomposition; criteria related to the system modeling; criteria related to the way of generating results and criteria in terms of evaluating results. Such ranking system permits a qualitative classification of the various approaches according to the specified criteria. Here, a boolean scale is sufficient to characterize the capabilities of the approaches. Firstly, the symbol $\ominus/\oplus$ is used to tag the methods not adapted/well adapted, incomplete/complete with respect to the considered criterion (table 2.2). They state a negative/incomplete ($\ominus$) or positive/complete ($\oplus$) tendency of the approaches with respect to the given criteria. They are defined in such a way that the optimal method would never be assigned the symbol($\ominus$). Secondly, in case the information contained in the articles do not enable the assessment of a criterion, symbol (?) is used. Finally, the symbol ($\odot$) means criteria that have no meaning for the method and are simply not applicable.

Symbols	Criteria
$\ominus$	Not adapted/Incomplete
$\oplus$	Well adapted/Complete
?	Not appreciable
$\odot$	No meaning/Not applicable

Table 2.2: Symbols used to characterize the approaches.

For example, distinguishing redundant and conflicting constraints is a criteria for evaluating numerical detection methods. However, there is no meaning applying it to evaluate structural detection methods since the latter only generate structural over-constraints. Of course, synthesis results are from our understanding of the publications.

2.3.1 Criteria attached to the level of detecting over-constraints

The first criterion is relative to the type of geometric over-constraints (Figure 2.18,a), which are either numerical (a⊕) or structural (a⊖). Second criterion concentrates on distinguishing redundant and conflicting constraints detected by numerical methods (Figure 2.18,b). Finally, in engineering design, designers could better debug and modify a geometric over-constraint if its spanning group is informed (Figure 2.18,c). This criterion evaluates numerical methods only.

Detection level		Gradation of criteria	
Level	Criteria	⊕	⊖
a	type	numerical	structural
b	redundant/conflicting	yes	no
c	spanning group	yes	no

Table 2.3: Criteria attached to the detection level (set 1)

2.3.2 Criteria related to the system decomposition

Decomposition is an important phase in geometric constraints solving domain. A large system is decomposed into small solvable subsystems which speed up the solving process. A desirable method should return the decomposition result to a user for debugging purpose by generating over-constrained components, which helps him/her locating the geometric over-constraints (d⊕). Also, the ability to generate rigid subsystems should be considered. Here, the *rigid* are of two meanings. Numerical methods detect the rigid subsystem which is solvable (finite solutions, e⊕) while structural methods detect the rigid subsystem which is structurally well-constrained (**Definition 5**, e⊖). Usually, the rigid subsystems are arranged with solving order and over-constraints with each subsystem can be detected by analyzing the subsystem individually.

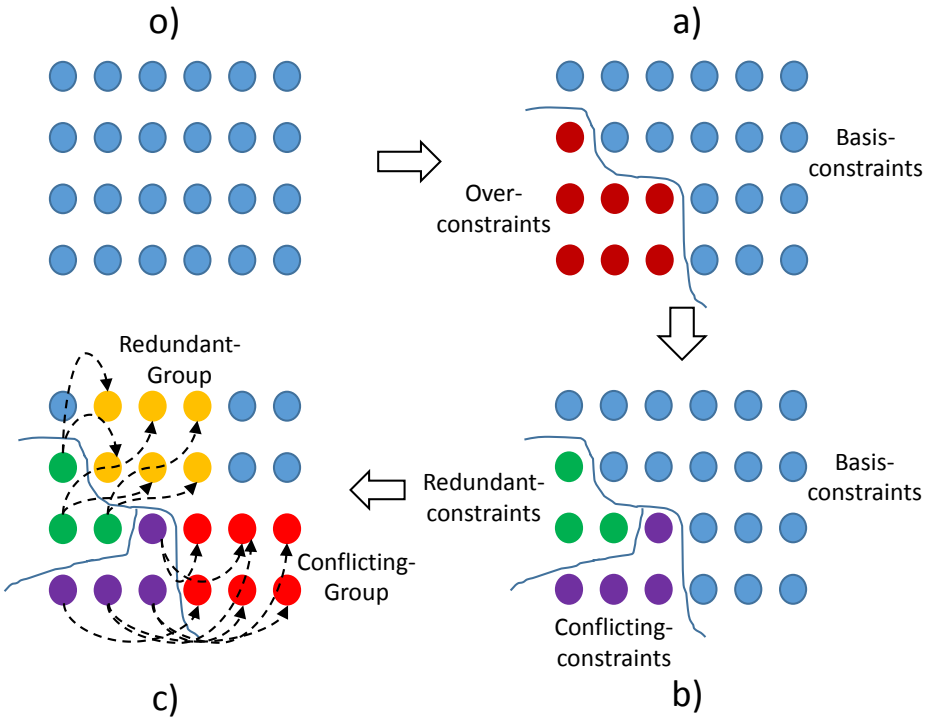


Figure 2.18: (o): Level o (a): Level a (b): Level b (c): Level c

.

Decomposition		Gradation of criteria	
Level	Criteria	$\oplus$	$\ominus$
d	over-constrained components	yes	no
e	rigid subsystems	numerical	structural
f	singular configuration	yes	no

Table 2.4: Criteria related to system decomposition (set 2)

Decomposition methods should take into account the singularities. Indeed, many methods work under a genericity hypothesis and decompose systems into generically solvable components. A generic configuration remains rigid (non-rigid) before and after an infinitesimal perturbation (*Combinatorial Rigidity*). A singular configuration, however, transforms from rigid (non-rigid) to non-rigid (rigid) after an infinitesimal perturbation. It happens when geometric elements are drawn with unspecified properties (collinearity, coplanarity, etc.). It may be the case that a solution of a decomposed system lies into a singular variety, e.g., includes some unspecified collinearity or coplanarity. In this case, it happens that the generically solvable components are no more solvable. For instance, the double-banana system (figure 2.36)

is generically over-constrained but becomes under-constrained if the height of both bananas is the same since the two "bananas" can fold continuously along the line passing through their extremities. Moreover, Jacobian matrix at singular configuration is rank deficiency, which introduces dependences between constraints. For example, the Jacobian matrix of the subsystem $\{p3, l2, c, c7, c8, c13\}$ of figure 2.19 is of size 7×7. But its rank is 5. The configuration is singular. Obviously, there is no redundant constraints and the singularity comes from the tangent constraints between c and $l1$, $l2$ (Xiaobo et al., 2002).

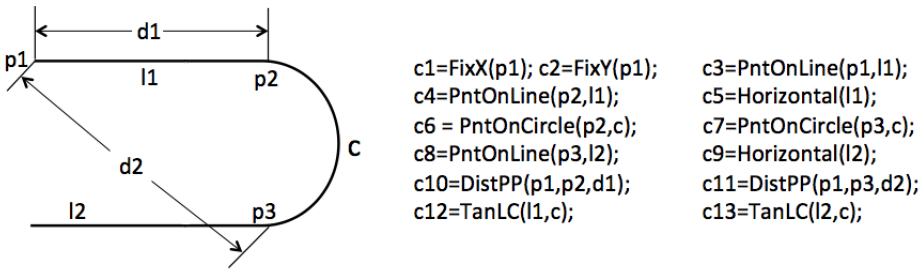


Figure 2.19: Singular configuration as described in (Xiaobo et al., 2002)

2.3.3 Criteria related to system modeling

This set of criteria characterize detection approaches with respect to system modeling: the type of geometries (g) and constraints (h), modeling at the level of equation or geometry (i), 3D or 2D space (j). The first criterion characterizes the type of geometries. Currently, geometric entities are either Euler geometries (g$\ominus$) such as line segments, cylinders, spheres etc or NURBS geometries (g$\oplus$). The second criterion deals with linear (h$\ominus$) and non-linear (h$\oplus$) constraints. The third criterion describes a system either at the level of equation (i$\oplus$) or geometry (i$\ominus$). Finally, a modeling system can either be in 2D (j$\ominus$) or 3D (j$\oplus$) space.

System modeling		Gradation of criteria	
Level	Criteria	$\oplus$	$\ominus$
g	geometries	Free-form	Euler
h	constraints	non-Linear	linear
i	modeling	equation	geometry
j	dimension	3D	2D

Table 2.5: Criteria related to system modeling (set 3)

2.3.4 Criteria related to the way of generating results

In reality, a designer may require that a modeler outputs geometric over-constraints iteratively when modeling a geometric system interactively. Iteratively means the method enables to generate results through steps/loops (k⊖) while single-pass methods generate the results all at once (k⊕). Also, a user-friendly method should enable the treatment of results for debugging purpose (m⊕). That is, locate the results at the level of geometries so that users can modify/remove them.

Results generation		Gradation of criteria	
Level	Criteria	⊕	⊖
k	way of detection	single-pass	iteratively
l	debugging	yes	no

Table 2.6: Criteria related to the way of generating results (set 4)

2.4 Detection approaches

Now that the criteria used to evaluate the different approaches have been introduced, this state-of-the-art gathers together existing techniques that are capable of detecting geometric over-constraints. The techniques are classified with respect to Definitions in section 2.2.1. The table 2.7 gathers together the results of this analysis.

2.4.1 Methods working at the level of geometry

This group of methods detect geometric over-constraints based on Dof analysis. Since these methods operate geometric entities directly, geometric information of the over-constraints are retained and thus easy to interpret.

Methods corresponding to the Definition 1

Fudos and Hoffman (Fudos and Hoffmann, 1997) introduced a constructive approach to solve a constraint graph, where geometric entities are lines and points and geometric constraints are distances and angles. In their reduction algorithm, triangles are found and merged recursively until the initial graph is rewritten into a final graph. The structurally over-constrained system/subsystem are detected in two ways. Firstly, before finding triangles, the approach checks if the subgraph is structurally over-constrained. Secondly, if a 4-cycle graph is met during the reduction process, then the system

is structurally over-constrained. A 4-cycle graph corresponds to two clusters sharing two geometric elements, which is structurally over-constrained.

Results of evaluating the method are as following:

- **Criteria set 1** Although the method allows for checking the constrained status of a system, it does not specify how to find the structural over-constraints as well as finding the spanning groups (a,c?). Since the method is structural, it is meaningless addressing how to distinguish redundant and conflicting constraints (b⊙).
- **Criteria set 2** The method enables to identify a 4-cycle graph which is structurally over-constrained component (d⊕). Also, the triangles found during the recursive process are the rigid subsystems (e⊖). Regarding to dealing with singular configurations, it is not mention in the original paper (f?).
- **Criteria set 3** Normally, the constraint systems are composed of Euler geometries (g⊖) with non-linear constraints (distances, angles h⊕) and modeled at the geometric level(i⊖) in 2D space (j⊖).
- **Criteria set 4** Since detecting geometric over-constraints are not addressed, there is no meaning discussing how the over-constraints are generated (k⊙) as well as debugging them(l⊙).

Methods corresponding to the Definition 5

Hoffman et al adapted their *Dense* algorithm (Hoffmann, Lomonosov, and Sitharam, 1998) to locate 1-overconstrained subgraph (satisfying $DOCs > DOFs - D + 1$) of 1-overconstrained graph (Hoffmann, Sitharam, and Yuan, 2004). The algorithm is composed of four main steps.

1. overloads the capacity from one arc from the source to a constraint by $D + 2$.
2. distributes a maximum flow in the overloaded network.
3. finds subgraph of density $\geq -D + 1$, where the density of a subgraph $A : d(A) = DOCs(A) - DOFs(A)$.
4. locates a minimal 1-overconstrained subgraph by deleting vertices one by one.

As it is shown in figure 2.20, generally locating a minimal subgraph of density $-D + 1$ is done as follows: first, by distributing an flow of weight

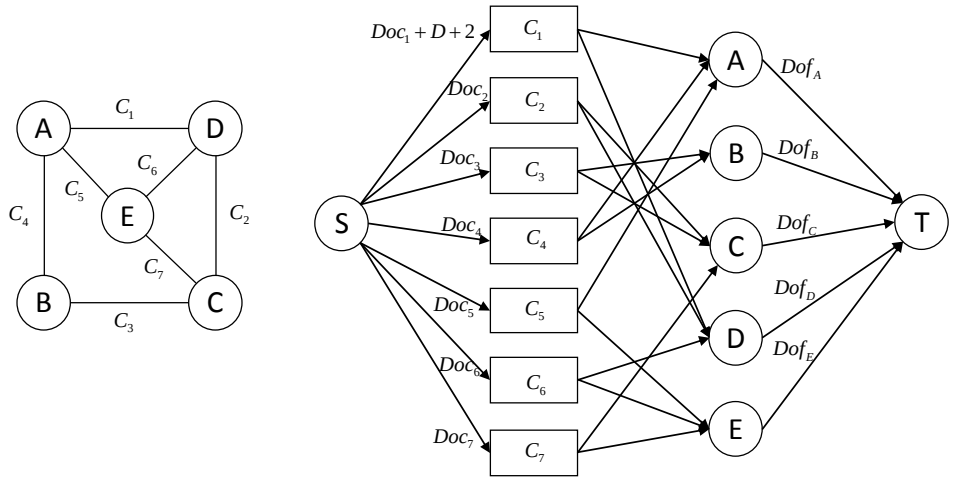


Figure 2.20: Left: Constraint graph with 5 entities and 7 constraints. Right: Flow network derived from the bipartite graph, where source S is linked to each constraint (capacity correspond to Doc_i of a constraint i) and each entity is linked to the sink T (capacity correspond to Dof of an entity)

.

$Doc_i + D + 2$ from each constraint to its end points(entities) to find a subgraph of density $-D + 1$. Such dense graph is found when there exists an edge whose edge cannot be distributed even with redistribution (Hoffmann, Lomonosov, and Sitharam, 1997). The algorithm continues to locate minimal 1-overconstrained subgraph but in our opinion, to check if whether a system is over-constrained or not, it is sufficient that the algorithm terminates at step 3. The authors suggest to further extend the algorithm to incrementally detect k-Overconstrained graphs. The algorithm allows for updating constraints efficiently and maintaining dynamically. Once the constraints have been identified, they are removed. The algorithm excludes large geometric structures that have rotational symmetry, however.

Results of evaluating the method are:

- **Criteria set 1** The method does not specify neither detecting geometric over-constraints nor the spanning groups (a,c?). Since the method is structural, talking about distinguishing redundant and conflicting constraints is meaningless (b⊙).

- **Criteria set 2** The algorithm locates the 1-overconstrained subgraph (d⊕) rather than rigid subsystems (e⊙). Regarding singular analysis of a system, it is not addressed by the method (f?).

- **Criteria set 3** The evaluation of this set of criteria on the method is the same with the previous's one except that the modeling dimension can be both 2D and 3D (j⊕⊖).

- **Criteria set 4** Since detecting geometric over-constraints are not addressed, there is no meaning discussing how the over-constraints are generated (k⊙) as well as debugging them (l⊙).

Methods corresponding to the Definition 6

Hoffmann's algorithm cannot deal with constraints such as alignments, incidences and parallelisms either generic or non-generic. Based on their work, Jermann et al (Jermann, Neveu, and Trombettoni, 2003) proposed the *Over-rigid* algorithm with the following modifications:

1. The overload is applied on a virtual node R (figure 2.21, figure 2.22) whereas in the *Dense* algorithm, it is applied on an constraint node.

2. The overload is $Dor + 1$ and the computation of Dor depends on the subsets of constraint entities to which R is attached. For example, $Dor(A, B)$ ($\{A, B\}$ is the subset of the configuration in figure 2.21) is different from $Dor(C, D, E)$ ($\{C, D, E\}$ is the subset of the configuration in figure 2.22). But in the *Dense* algorithm, the overload is invariant with different subsystems.

3. The R node is attached to *Dor-minimal* subsets of objects in order to find over-rigid subsystems.

The Dor varies with different subsystems. Readers can refer to the original paper to know the computation details. The *Over-rigid* algorithm is initially designed to check whether a system is structurally well-constrained or not. However, the authors do not show in specific how to detect structurally over-constrained systems as Hoffmann et al did to the *Dense* algorithm. Since the algorithm modified the *Dense* algorithm, the algorithm can be adapted to detect over-constrained systems if setting the overload to $Dor + 2$ and following the step 1-3 of the *Dense* algorithm (section 2.4.1). The evaluation of adapted version of *Over-rigid* algorithm is the same as the modified version of *Dense* algorithm.

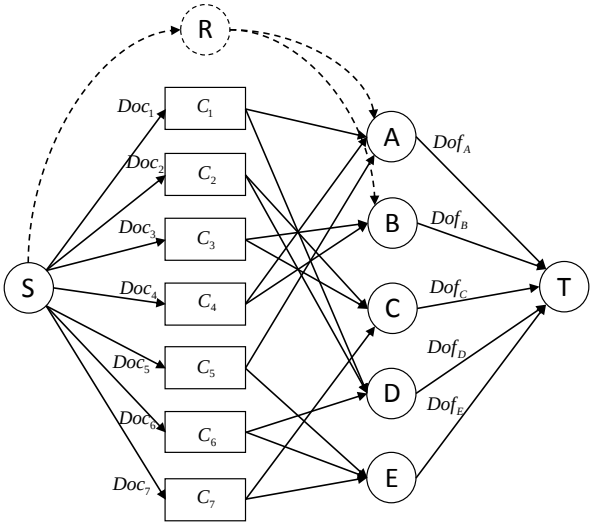


Figure 2.21: Overloading to subset {A,B}

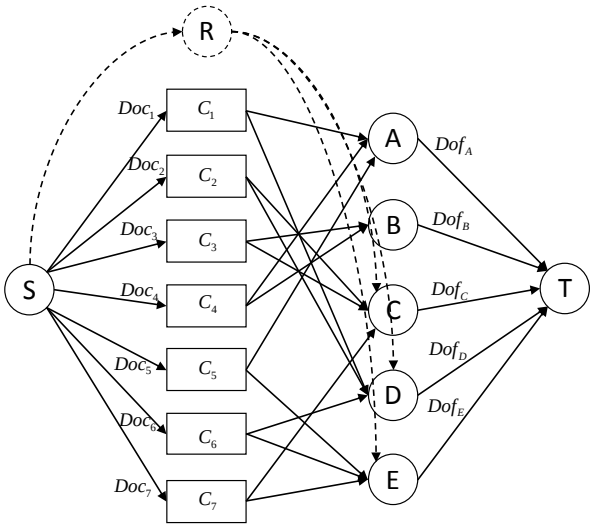


Figure 2.22: Overloading to subset {C,D,E}

Methods corresponding to the Definition 7

Latham et al (Latham and Middleditch, 1996) detected over-constrained subgraphs based on DoFs analysis by finding a maximum weighted matching (MWM) of a bipartite graph. The method decomposes the graph into minimal connected components which they called balanced sets. If a bal-

anced set is in a predefined set of patterns, then the subproblem is solved by a geometric construction, otherwise a numeric solution is attempted. The method addresses symbolic constraints and enables to identify under- and over-constrained configurations.

As it is shown in figure 2.23, constraints system is initially represented with a constraint graph with two classes of nodes representing DoFs of geometric entities and constraints respectively. Then, it is transformed into a directed graph by specifying directions from constraints nodes to entities nodes. After that, maximum matching between constraints and entities is applied and those unsaturated constraints nodes are geometric over-constraints. Moreover, they addressed the over-constrained problems by prioritizing the given constraints, where over-constraints can automatically be corrected using constraint priorities.

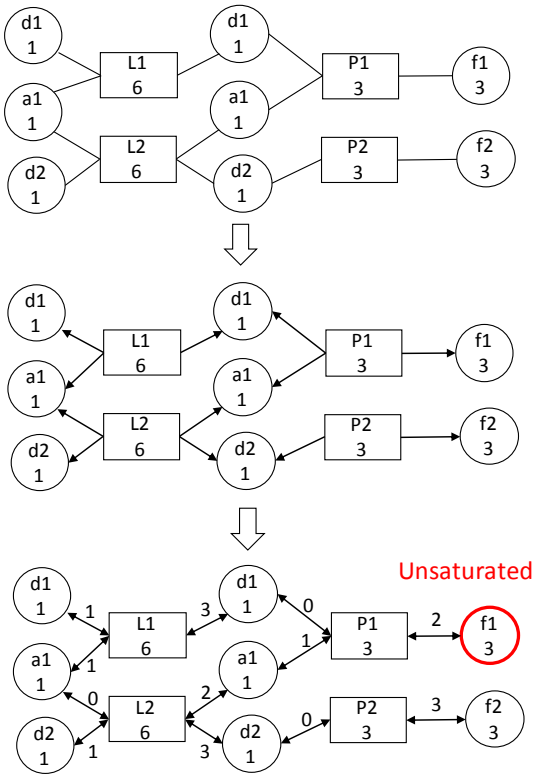


Figure 2.23: Over-constraints detection process of an example taken from (Latham and Middleditch, 1996). The unsaturated node is the geometric over-constraint.

.

Results of evaluating the method are:

- **Criteria set 1** The unsaturated constraints are geometric over- constraints (a⊖). Since the detected over-constraints are structural, there is no meaning discussing redundant and conflicting constraints as well as the spanning groups (b,c⊙).

- **Criteria set 2** The subgraph containing an unsaturated constraint node is the over-constrained component (d⊕). It can be found by tracing the descendant nodes of the unsaturated n ode. Moreover, the algorithm enables a decomposition of the system into balanced subsets which are the rigid subsystems (e⊖). Also, analyzing the singular configurations is not discussed by the authors (f?).

- **Criteria set 3** The results of evaluation of this set of criteria are the same with those of *Over-rigid* algorithm except the whole system is modeled in 3D space (j⊕).

- **Criteria set 4** The over-constraints are detected in single-pass way (k⊕). And they proposed to correct the constraints according to constraints priorities (l⊕).

2.4.2 Methods working at the level of equation

In general, almost all the geometric constraints can be translated mechanically into a set of algebraic equations (Hoffmann, Lomonosov, and Sitharam, 1998). Therefore, detecting geometric over-constraints is equivalent with identifying a set of conflicting/redundant equations. However, even if detection works at the level of equations, the treatment is to be done at the level of geometries and constraints.

Methods corresponding to the Definition 8

A variation of Latham's method directly deals with algebraic constraints, where a maximum cardinality of bipartite matching is used. D-M algorithm decomposes an equations system into smaller subsystems by transforming the equation system into a bipartite graph and canonically decomposes the bipartite graph through maximum matchings and minimum vertex covers. It decomposes a system into over-constrained, well-constrained and under-constrained subsystems (Dulmage and Mendelsohn, 1958). It has been used for debugging in equation-based modeling systems such as Modelica (Bunus and Fritzson, 2002a). Serrano has been interested in using graph-theoretic algorithm to prevent over-constrained systems where all constraints and geometric entities are of DoF one (Serrano, 1987).

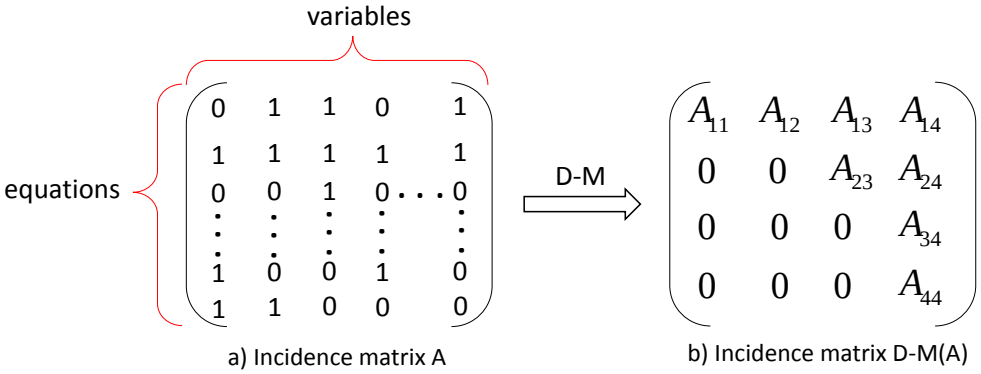


Figure 2.24: D-M decomposition of incidence matrix A.

The process of D-M decomposition: D-M(A) = $A(p, q)$ does not require A need to be square or full structural rank. $A(p, q)$ is split into a 4-by-4 set of coarse blocks: where $A12$, $A23$, and $A34$ are square with zero-free diagonals. The columns of $A11$ are the unmatched columns, and the rows of $A44$ are the unmatched rows. Any of these blocks can be empty. The whole decomposition is composed of coarse and fine decomposition.

Coarse decomposition

- $[A11A12]$ is the underdetermined part of the system—it is always rectangular and with more columns than rows, or does not exist.
- $A23$ is the well-determined part of the system—it is always square.
- $[A34; A44]$ is the overdetermined part of the system—it is always rectangular with more rows than columns, or does not exist.

Fine decomposition The above sub-matrices are further subdivided into block upper triangular form via the fine decomposition. Consequently, strong connected components are generated and linked with solving order (Ait-Aoudia, Jegou, and Michelucci, 2014). By analyzing each component following the solving order, the system is updated dynamically and over-constraints are generated iteratively.

Results of evaluating the method are:

- **Criteria set 1** Equations of $[A34; A44]$ are structural over-constraints (a⊖). Evaluation of criteria b and c is meaningless since the method is structural (b,c⊙).

- **Criteria set 2** [$A34$; $A44$] after coarse decomposition is the structural over -constrained subpart (d⊕). The strong connected components after fine decomposition are structural rigid subsystems (e⊖). The method does not discuss on analyzing singular configurations (f⊖).

- **Criteria set 3** Since the modeling is based on system of equations, any geometric constraints that are able to be transformed into system of equations can be analyzed by the method. Therefore, the results for evaluating this set of criteria are (g⊕⊖, h⊕⊖, i⊕, j⊕⊖).

- **Criteria set 4** Structural over-constraints are contained in O_G and output in a single-pass way along with generation of O_G (k⊕). The method does not discuss on debugging the over-constraints (l?).

Methods corresponding to the Definition 10

Linear methods In this section, we gather together the existing techniques from linear algebra that are capable of analyzing linear constraint systems. We consider linear constraint systems in the matrix form $Ax = b$, where A has dimension $m \times n$, and $n \geq m \geq r$ with r being the rank. The notation $A[i : j, l : k]$ defines the matrix obtained by slicing the ith to jth rows, and the lth to kth columns of A. According to (Strang, 2006), methods such as Gauss-Jordan Elimination, LU and QR Factorization present a good characteristic of locating inconsistent/redundant equations.

Gauss-Jordan Elimination with partial pivoting The elimination process is terminated once a reduced row echelon form is obtained (An example is shown in figure 2.25). Exchanging rows at the start of kth stage to ensure that:

$$\left|A_{kk}^{(k)}\right| = \max_{i \geqslant k} \left|A_{ik}^{(k)}\right| \tag{2.6}$$

where $A_{ik}^{(k)} = A[i, k]$, an element of ith row and kth column in A.

Numerical over-constraints are identified by searching lines that contain only 0. The vector b is updated to b_{new} when transforming $[A, b]$. The last $m - r$ values of the b_{new} allow to further distinguish redundant (equal to 0) and conflicting (not equal to 0) constraints.

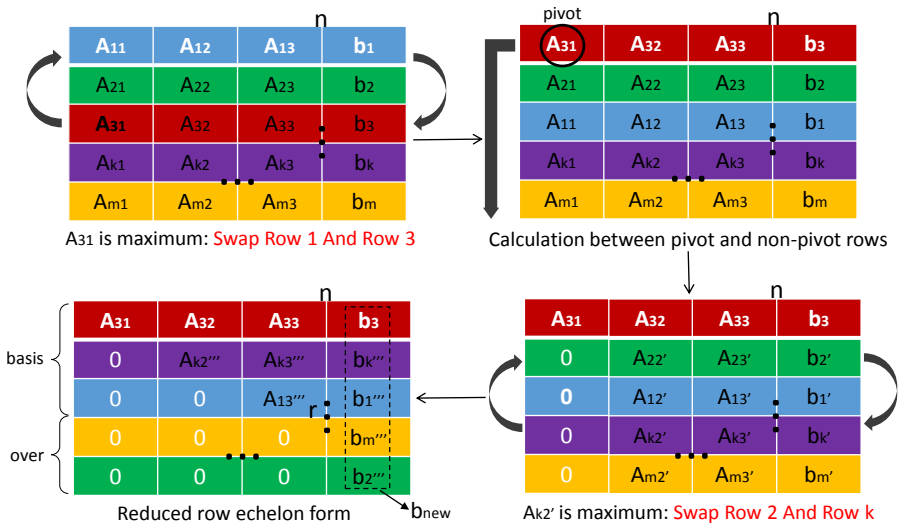


Figure 2.25: Gauss elimination with partial pivoting.

Results of evaluating the method are as follows:

- **Criteria set 1** The method allows for detecting redundant and conflicting constraints (a,b⊕). However, the method does not tell how to find spanning groups of an over-constraint (c?).
- **Criteria set 2** The method does not enable to decompose a system. There is no meaning evaluate the method with respect to system decomposition criteria (d,e,f⊙).
- **Criteria set 3** The method analyzes linear equations. Therefore, any geometry (g⊕⊖) with linear constraints (h⊖) in 3D or 2D space (j⊕⊖) modeling at equation level (i⊕) can be handled by the method.
- **Criteria set 4** The over-constraints are output all at once (k⊕) after detection without debugging them (l⊖).

In the work of (Light and Gossard, 1983), they used this method to detect invalid dimensioning schemes. Note that, in the following sections, G-J is short for Gauss-Jordan elimination with partial pivoting.

LU factorization with partial pivoting The method is a *high-level* algebraic description of G-J (Okunev and Johnson, 2005). The process is shown in figure 2.26, where P is the permutation matrix reordering the rows. The number of non-zero diagonal elements of U is the rank r. The

last $m-r$ rows of the reordered matrix $P*A$ corresponds to the numerical over-constraints.

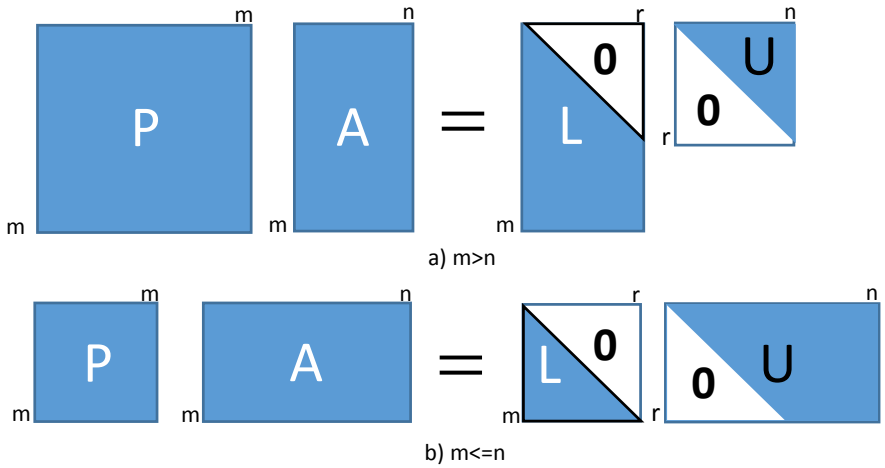


Figure 2.26: LU Factorization with partial pivoting

However, the factorization itself does not manipulate directly on b, which means that distinguishing redundant and conflicting constraints is unavailable. To know them, we need further extension:

$$\left.\begin{array}{c} Ax = b \\ PA = LU \end{array}\right\} Ux = L^{-1}Pb \tag{2.7}$$

Now the distinguish step is similar to the one of G-J. That is, by comparing the last $m-r$ elements of $L^{-1}Pb$ with 0, redundant and conflicting constraints can be distinguished. However, one has to notice that the deduction process is under the condition that L should be invertible.

To the best of our knowledge, using this method to detect geometric over-constraints in not convincingly demonstrated in literature. Evaluation of the method with respect to five criteria is the same as the one of G-J. Note that in the following sections, we use LU in short for LU factorization with partial pivoting.

QR Factorization with column pivoting Before applying QR , coefficients matrix A should be transposed first($A = A^t$) since QR operates on columns of a matrix. QR exchanges columns at the start of the kth stage to ensure that:

$$\left\| A_k^{(k)}(k:m)) \right\|_2 = \max_{j \geqslant k} \left\| A_j^{(k)}(k:m)) \right\|_2 \tag{2.8}$$

where $A_j^{(k)}(k:m) = A[k:m, j]$

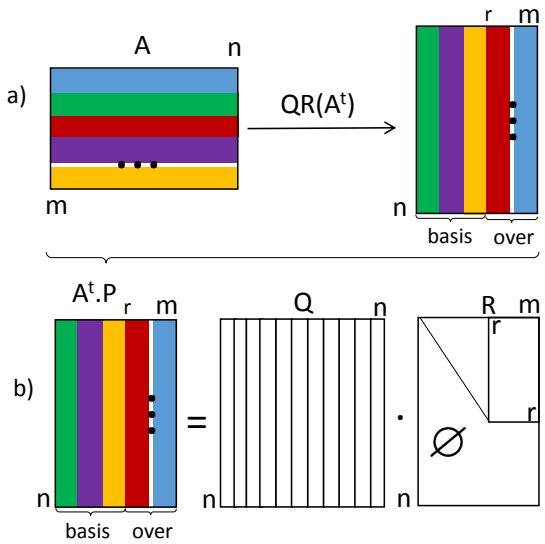


Figure 2.27: QR Factorization with column pivoting.

As shown in figure 2.27, P is the permutation matrix where the information about columns exchanges is stored. R is a triangular matrix where rank r is the number of non-zeros diagonal elements. Equations corresponding to $A^t.p[:, r+1:m]$ are the over-constraints (Dongarra and Supercomputing, 1990).

Similar to LU, further deduction is needed to distinguish redundant and conflicting constraints. First, the matrix $Q(:, 1:r)$ is inverted using the following equation:

$$A^t(:, 1:r) = Q(:, 1:r).R(1:r, 1:r) \tag{2.9}$$

and is then used in the following equation:

$$A^t(:, r+1:n) = Q(:, 1:r).R(1:r, r+1:n) \tag{2.10}$$

thus providing the following relationship between the two sliced matrices $A^t(:, r+1:n)$ and $A^t(:, 1:r)$:

$$A^t(:, r+1:n) = A^t(:, 1:r).R(1:r, 1:r)^{-1}R(1:r, r+1:n) \tag{2.11}$$

The relationship between over-constraints and independent constraints are revealed in the matrix $R(1:r, 1:r)^{-1}R(1:r, r+1:n)$ in equation 2.11. From the matrix, the spanning group of an over-constraint could also be known.

Finally, to identify the redundant and conflicting equations, the new b vector after factorization is redefined as follows:

$$b_{new} = b(r+1:n) - b(1:r).R(1:r,1:r)^{-1}R(1:r,r+1:n) \qquad (2.12)$$

Redundant and conflicting constraints can be further distinguished by comparing the value of the last $m-r$ elements of b_{new} with 0.

The method was adopted by Hu et al (Hu, Kleiner, and Pernot, 2017) to detect over-constraints of B-splines. Evaluation of the method with respect to five criteria is the same as the one of G-J. We use QR in short for the method for discussion in following sections.

Non-linear methods Detecting non-linear geometric constraints systems is more complicated than linear ones. Since non-linear detection methods include symbolic methods from abstract algebra, we introduce the mathematical fundamentals to make it easy to understand. The following two theorems are induced from (Cox, Little, and O'shea, 1992). Readers can find more details about concepts like ideals, affine varieties etc in the book.

Theorem 1 For a system of polynomial equations $f_0 = f_1 = \cdots = f_s = 0$, where $f_0, f_1, \cdots, f_s \in \mathbb{C}[x_1, \cdots, x_n]$; If affine variety $W(f_1, \cdots, f_s) \neq \varnothing$ while $W(f_0, f_1, \cdots, f_s) = \varnothing$, then $f_0 = 0$ is a conflicting equation; If $W(f_0, f_1, \cdots, f_s) = W(f_1, \cdots, f_s) \neq \varnothing$, then $f_0 = 0$ is a redundant equation; If $W(f_0, f_1, \cdots, f_s) \neq \varnothing$, $W(f_1, \cdots, f_s) \neq \varnothing$ and $W(f_0, f_1, \cdots, f_s) \neq W(f_1, \cdots, f_s)$, then $f_0 = 0$ is an independent equation.

Theorem 2 (**Hilbert′s weak Nullstellensatz**) Let k be an algebraically closed field. If $f, f_1, ..., f_s \in k[x_1, ..., x_n]$ are such that $f \in I(W(f_1, \cdots, f_s))$, then there exists an integer $m \geq 1$ such that $f^m \in \langle f_1, \cdots, f_s \rangle$ (and conversely).

Based on the Theorem 2, Michelucci et al (Michelucci and Foufou, 2006a) deduced the Corollary 1.

Corollary 1 Let k be an algebraically closed field and $W(f_1, \cdots, f_s) \neq \varnothing$. If $f, f_1, \cdots, f_s$ have the common root w, then rank$([f^{'}(w), f_1^{'}(w), \cdots, f_s^{'}(w)]^T) < s+1$.

Informally, Corollary 1 tells that if a system of polynomial equations containing redundant equations, then the Jacobian matrix of the equations at the affine space (solution space) must be row rank deficiency. However,

the reverse is not correct. In other words, if there exists Jacobian matrix whose rank is deficiency at the solution space, then system of polynomial equations does not necessarily contain redundant equations. A typical example is the singular configuration in section 2.3.2: the system is row rank deficiency at the solution space but the system does not contain geometric over-constraints. It is the singular configuration that causes the system rank deficiency. Therefore, to detect over-constraints through analyzing Jacobian matrix, one has to note that Jacobian matrix should be computed on configurations in the solution space but avoid the singular ones.

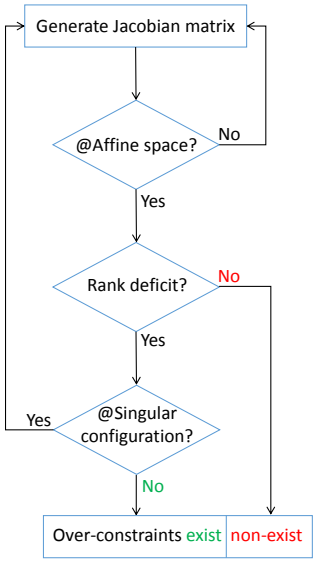


Figure 2.28: Over-constraints detection based on Jacobian matrix analysis

We propose a schema on determining the existence of over-constraints through analyzing Jacobian matrix in figure 2.28. That is, compute the Jacobian matrix at a configuration from affine space. If the rank is full, then there is no over-constraints. Otherwise, we check if the configuration is singular. If not, then there exists over-constraints otherwise we move to test the other configuration in the affine space. Loops means that one has to go back to generate Jacobian matrix at different points until the existence/non-existence of over-constraints can be determined. It is a recursive process of finding points that can be used to determine the existence/non-existence of over-constraints. In reality, however, affine space sometimes is hard to find or even does not exist. Moreover, the singularity of a configuration is sometimes difficult to test. A lot of research work have been done to address the two issues.

The first group of methods are symbolic algebraic methods, which compute a Grobner basis for the given system of equations. Algorithms to compute these bases include those by Buchberger (Bose, 1995), and by Wu-Ritt (Chou, 1988b; Wu, 2012). These methods, essentially, transform the system of polynomial equations into a triangular system whose solutions are those of the given system.

Grobner Basis(GB) Assume a set of polynomials $f_0, f_1, \cdots, f_s \in \mathbb{C}[x_1, \cdots, x_n]$. The reduced Grobner basis(rgb_0) of the ideal $\langle f_1, \cdots, f_s \rangle$ satisfies $rgb_0 \neq \{1\}$ and $rgb_0 \neq \{0\}$ with respect to any ordering. The new reduced Grobner basis of the ideal $\langle f_0, f_1, \cdots, f_s \rangle$ is rgb_{new}. If $rgb_{new} = \{1\}$, $f_0 = 0$ is a conflicting equation; if $rgb_{new} \equiv rgb_{old}$, $f_0 = 0$ is a redundant equation(b$\oplus$); if $rgb_{old} \subset rgb_{new}$, $f_0 = 0$ is an independent equation (Cox, Little, and O'Shea, 2015).

Results of evaluating the method are as follows:

- **Criteria set 1** Obviously, the above method can tell if a constraint is redundant or conflicting (a,b$\oplus$). However, the method does not support finding spanning group of a (c$\ominus$).

- **Criteria set 2** The method is initially designated for solving polynomial equations. Therefore, there is no meaning evaluating it with set of criteria on system decomposition (d,e$\odot$). The method does not analyze the singularity of a configuration (f$\ominus$).

- **Criteria set 3** The method analyzes non-linear equations. Therefore, any geometries (g$\oplus\ominus$) with non-linear constraints (h$\oplus$) in 3D or 2D space (j$\oplus\ominus$) modeling at equation level (i$\oplus$) are applicable for the method.

- **Criteria set 4** To detect a set of over-constraints, the process of computing reduced grobner basis requires inputing one equation at a time. Therefore, the over-constraints are not generated all at once but iteratively (k$\ominus$). However, debugging these over-constraints are not supported (l$\ominus$).

Construction of a Grobner basis is a potentially time-consuming process. Hoffman used this technique to do geometric reasoning between geometric configurations (Hoffmann, 1989). In terms of detecting geometric over-constraints, Kondo (Kondo, 1992) initially used Grobner basis method to test dependencies among 2D dimension constraints.

Wu-Ritt characteristic sets Let a system of polynomial equations $P = \{f_0 = f_1 = \cdots = f_s = 0\}$, where $f_0, f_1, \cdots, f_s \in \mathbb{Q}[x_1, \cdots, x_n]$ represent system of constraints and Zero(P) denote the set of all common zeros of $\{f_0, f_1, \cdots, f_s\}$. The system contains redundant equations iff there exist a polynomial p such that:

$$Zero\,(P - \{p\}) \equiv Zero\,(P) \tag{2.13}$$

For the polynomial set P, its zero set can be decomposed into a union of zero sets of polynomial sets in triangular form through *Wu-Ritt's zero decomposition algorithm*:

$$Zero\,(P) = \bigcup_{1 \leqslant i \leqslant k} Zero\,(TS\,\{i\}\,/I\,\{i\}) \tag{2.14}$$

where each $TS\{i\}$ is a polynomial set in triangular form, $I\{i\}$ is the production of the initials of the polynomials in $TS\{i\}$ and k is the number of zero sets. The system contains inconsistent equations iff $k \equiv 0$ (Chou, 1988a). In the work of Gao and Chou (Gao and Chou, 1998), they present complete methods for identifying conflicting and redundant constraints based on Wu-Ritt's decomposition algorithm. Also, the algorithm is used to solve Pappus problems on deciding if a configuration can be drawn with ruler and compass.

Results of evaluating Gao's method are as follows:

- **Criteria set 1** As discussed above, their method enable to detect conflicting and redundant constraints (a,b⊕) but cannot find the spanning groups (c⊖).
- **Criteria set 2** The method decomposes a set of polynomials into a union of zero sets in triangular form. No over-constrained subparts, rigid subsystems are generated as well as singular configurations are analyzed (d,e,f⊖).
- **Criteria set 3** The method analyzes non-linear equations. Any geometries (g⊕⊖) with non-linear constraints (h⊕) in 3D or 2D space (j⊕⊖) modeling at equation level (i⊕) are applicable.
- **Criteria set 4** The results are the same as those of evaluating Grobner basis.

Symbolic detection methods are sound in theory but suffer from high computation cost. As discussed previously, the worst-case can be doubly exponential. Moreover, the reduced Grobner basis has to be computed every

time an equation is to be analyzed. Therefore, this method are not able to deal with large systems of equations.

The second group of methods analyze Jacobian matrix of equation systems. Contrary to symbolic methods, these numerical methods are more practical in computation but are theoretical deficiency in some cases. On one hand, if the affine space of a system does not exist, an equivalent one that sharing similar Jacobian structure need to be found. On the other hand, even if the Jacobian matrix of a configuration is row rank deficiency in affine space, it has to be sure that the configuration is not singular.

Perturbation method Haug proposed a perturbation method to deal with singular configurations and detect redundant constraints in mechanical systems (Haug, 1989). Initially, assume system of equations $\Phi(q) = 0$ and the corresponding Jacobian matrix Φ_q is rank deficiency at q. As we discussed before, this is not enough to determine the existence of the over-constraints since the singular configuration always make a Jacobian matrix rank deficiency. He suggested to analyze Jacobian matrix at one more configuration by doing the following:

- Add a small perturbation δq to q and obtain $\Phi_q \delta q = 0$. This process is based on the Implicit Function Theorem (Krantz and Parks, 2012).

- Applying G-J elimination to Φ_q , $\Phi_q \delta q = 0$ is transformed into $\begin{bmatrix} \Phi_u^I & \Phi_v^I \\ 0 & \Phi_v^R \end{bmatrix} \begin{bmatrix} \delta u \\ \delta v \end{bmatrix} = 0$. Φ_u^I is the upper triangular matrix with 1s as diagonal elements. Φ_v^R can be treated as the matrix with all 0s under given tolerance. Equations in $\Phi(q) = 0$ corresponding to Φ_u^I part: $\Phi^I(q) = 0$ are independent.

- Now, $\Phi_q \delta q = 0$ can be simplified into $\Phi_u^I \delta u + \Phi_v^I \delta v = 0$ and thus $\delta v = -(\Phi_v^I)^{-1} \Phi_u^I \delta u$, $\delta q = \begin{bmatrix} \delta u \\ \delta v \end{bmatrix} = \begin{bmatrix} \delta u \\ -(\Phi_v^I)^{-1} \Phi_u^I \delta u \end{bmatrix}$

- Assume q is perturbed to new point q^* satisfying $q^* = q + \delta q$. To ascertain it lies in the affine space, is should satisfy $\Phi(q*) = 0$. This is equivalent to $\Phi^I(q*) = 0$ since the latter is composed of all independent equations of the former.

- Solving $\Phi^I(q*) = 0$, $q^* = q + \delta q$, $\delta q = \begin{bmatrix} \delta u \\ \delta v \end{bmatrix} = \begin{bmatrix} \delta u \\ -(\Phi_v^I)^{-1} \Phi_u^I \delta u \end{bmatrix}$, the value of q^* is obtained.

- Compute the rank of Jacobian matrix at q^*: Φ_{q^*} and check if it is rank deficiency.

We can see from above that obtaining an appropriate value of the perturbation δq so that q^* lies in the affine space is the main part of the work.

Results of evaluating the method are as follows:

- **Criteria set 1** The method enables to detect geometric over-constraints (a$\oplus$) but does not distinguish redundant and conflicting constraints (b$\ominus$). Finding the spanning groups is also not supported (c$\ominus$).
- **Criteria set 2** The method mainly detects the over-constraints based on analyzing the Jacobian matrix of a whole system. There is no meaning evaluate the method with respect to system decomposition criteria (d,e,f$\odot$).
- **Criteria set 3** The method analyzes both linear and non-linear equation systems. Therefore, any geometries (g$\oplus\ominus$) with non-linear and linear constraints (h$\oplus\ominus$) in 3D or 2D space (j$\oplus\ominus$) modeling at equation level (i$\oplus$) are applicable for the method.
- **Criteria set 4** The over-constraints are generated in a single-pass way since Jacobian matrix analysis is on the whole system at one time (k$\oplus$). However, debugging the over-constraints is not addressed (l$\ominus$).

His method selects two points in affine space to determine the existence of geometric over-constraints. If Jacobian matrix at any point is full rank, then there is no over-constraint. However, if the rank of Jacobian matrix at both points is deficiency, then there exists geometric over-constraints.

Numerical Probabilistic Method (NPM) Roots of system of equations can be sometimes hard to find or even do not exist. In these cases, the affine space does not exist. Sebti Foufou et al (Foufou, Michelucci, and Jurzak, 2005) suggest a Numerical Probabilistic Method (NPM), which is to test Jacobian matrix at random points instead of the affine space. However, there is a risk that Jacobian matrix is row rank deficiency at the chosen points and the corresponding configurations happen to be singular. They suggest to test on more testing points to reduce the possibility of happening such case. Moreover, in order to get more confidence, authors suggest that testing at 10 different points should be sufficient. NPM is practical in computation but is not sound in theory since the testing points are not necessarily all in affine space.

Results of evaluating the method are as follows:

- **Criteria set 1** The method enables to identify numerical over-constraints (a⊕). However, it can neither distinguish redundant and conflicting constraints nor finding the spanning group of an over-constraint (b,c⊖).

- **Criteria set 2** The method can be used to decompose a system into rigid subsystems (e⊖). However, decomposition into over-constrained components as well as analyzing singular configurations are not supported (d,f⊖).

- **Criteria set 3** The method analyzes both linear and non-linear equation systems. Therefore, any geometries (g⊕⊖) with non-linear and linear constraints (h⊕⊖) in 3D or 2D space (j⊕⊖) modeling at equation level (i⊕) are applicable for the method.

- **Criteria set 4** Numerical over-constraints are detected all at once (k⊕) but debugging them are not supported (l⊖).

Witness Configuration Method (WCM) Instead of randomly selecting configurations, Michelucci et al. suggested to study the Jacobian structure at witness configurations where incidence constraints are satisfied (Michelucci and Foufou, 2006b). The witness configuration and the target configuration shares the same Jacobian structure, which is non-singular in affine space. As a consequence, all the numerical over-constraints can be identified (Michelucci and Foufou, 2006a). More recently, Moinet et al. developed tools to identify conflicting constraints through analyzing the witness of a linearized system of equations (Moinet, Mandil, and Serre, 2014). Their approach has been successfully applied to the well-known Double-Banana geometry.

For a geometric constraints system represented with a set of equations $F(U, X) = 0$, where U denotes a set of parameters with prescribed values U_T(T for target), and X is the vector of unknowns. The solution is denoted as X_T. A witness is a couple (U_W, X_W) such that $F(U_W, X_W) = 0$. Most of the time, U_W and X_W are different from U_T and X_T respectively. The witness (U_W, X_W) is not the solution but shares the same combinatorial features with the target (U_T, X_T), even if the witness and the target lie on two distinct connected components of the solution set. Therefore, analyzing a witness can detect numerical over-constraints of a system(Michelucci et al., 2010; Michelucci et al., 2006). These numerical over-constraints are not only the structural ones but also geometric redundancies.

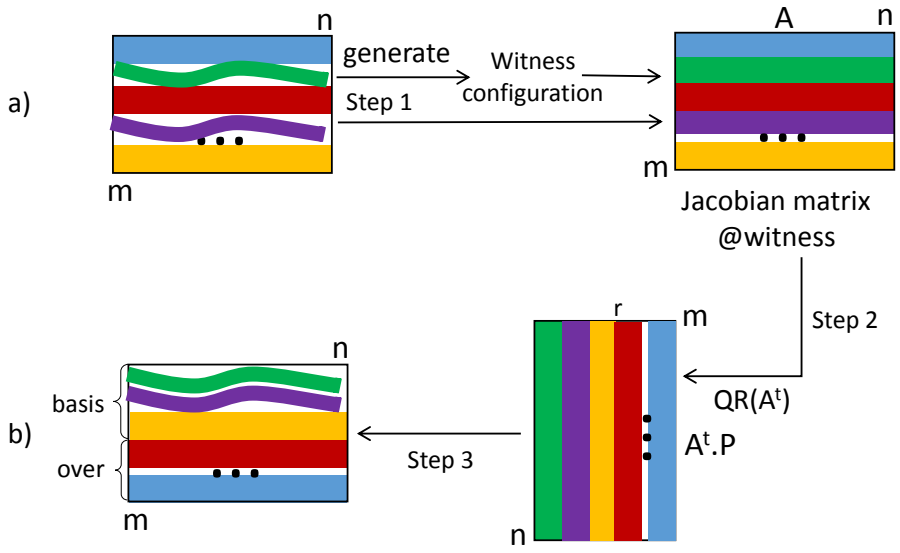


Figure 2.29: Witness configuration method

Figure 2.29 shows the witness method combining QR for detection. Step one aims at generating the witness configuration while at step two, QR is applied on the Jacobian matrix A. As a result, the rows of equations are re-ordered by P and the number of basis constraints is revealed by r. Finally, coming back to the re-ordered original equations, the first r equations are the basis constraints while the remaining ones are the numerical over-constraints. Note that, QR can be replaced with G-J in the process and would generate result different from the one of QR since the two methods adopt different sorting rows strategy.

The results of evaluating the method are the same as those of evaluating NPM (section 2.4.2) except that the property of **Correct** is retained (m⊕). Michelucci et al (Michelucci and Foufou, 2006a) have proved that the WCM can identify all the dependencies among constraints. In other words, if removing these dependent constraints, the remaining constraints are independent.

WCM Extension Thierry et al (Thierry et al., 2011) extend WCM to incrementally detect over-constrainedness and thus to compute a well-constrained boundary system. Also, they design the so called *W-decomposition* to identify all well-constrained subsystems, which manages to decompose systems non-decomposable by classic combinatorial methods.

Results of evaluating the method are as follows:

- **Criteria set 1** The results of evaluation within this set of criteria are the same with those of NPM.

- **Criteria set 2** The *W-decomposition* enables to efficiently identify the maximal well-constrained subsystems of an articulated system as well as further decompose a rigid system into well-constrained subsystems (e⊖) but finding over-constrained components is not discussed (d?). For finding the spanning groups, it is not supported (f⊖).

- **Criteria set 3** The results are the same with those of evaluating NPM.

- **Criteria set 4** Working on the witness, the naive idea would be to try and remove constraints one by one and, at each step, compute the rank again to determine if the constraint is redundant with the remaining set. However, the author points out that performing this way is expensive. They consider an incremental construction of the geometric constraint system to identify the set of redundant constraints with no additional costs in comparison to the basic detection of redundancy (k⊖). The method does not support debugging over-constraints (l?).

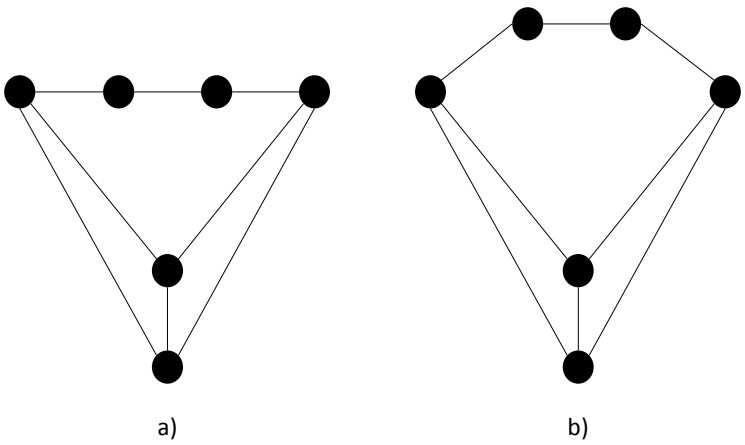


Figure 2.30: a) Rigid sketch b) Non-rigid sketch (Thierry et al., 2011)

.

Generating a witness configuration Sometimes, when certain geometric elements happens to be drawn with specific properties (collinearity, coplanarities, etc) without representing a real constraint, the sketch is not typical of the expected solution, for example it does not satisfy incidence

constraints. Thus cannot be used as a witness configuration. A witness configuration should be generic when it remains rigid before and after infinitesimal perturbation. Likewise, if the sketch is not rigid before perturbation, it should be not rigid after the perturbation (*Combinatorial Rigidity*). For example, figure 2.30-a) is not generic: a small perturbation on the dimensions of the bars will result in a non-rigid sketch shown in figure 2.30-b). However, figure 2.30-b) is generic: if a small perturbation is introduced in the dimension of the sketch, it will remain non-rigid. Usually, non-generic sketches are constituted with aligned line segments presented in figure 2.30-a). The collinearity will induce artificial redundancy between the constraints associated with the collinear vectors. As a result, before using the WCM, one has to make sure the witness configuration is typical of the expected solution.

Here, we adapted the algorithm of Moinet (Moinet, Mandil, and Serre, 2014) for generating generic witness configurations. Other methods for generating witness configurations can be found in (Thierry et al., 2011; Kubicki, Michelucci, and Foufou, 2014). Moinet's algorithm contains the following steps:

1. Compute the Jacobian matrix of system of equations.
2. Calculate the rank r_{old} of Jacobian matrix at initial sketch.
3. Randomly perturb the initial sketch (usually generated by users), regenerate the Jacobian matrix, and recompute the rank r_{new} at the new position(new sketch).
4. If $r_{new} > r_{old}$, replace the initial sketch by the new one and reiterate the third step.
5. Otherwise the old sketch is generic.

2.4.3 Hybrid methods

Serrano's Serrano analyzes systems of equations (h$\oplus$) to select a well constrained, solvable subsets from candidate constraints(Serrano, 1987). His method first detects structural over-constraints (a$\ominus$) if there are equations uncovered after maximum matching. To further detect numerical over- constraints (a$\oplus$) within strong connected components (e$\ominus$), symbolic and numerical method are used. The symbolic method used is pure symbolic operations, where constraints are eliminated one by one by substituting one variable into other equations until a final expression is obtained. Also, nonlinear equations are linearized and G-J elimination method is applied to analyze them. He repeated the above process until finally redundant and

conflicting constraints are all distinguished (b⊕). Moreover, the author suggests the spanning group of an over-constraint is a set of constraints within the same strong connected component (c⊕). However, it will generate wrong results if linearize non-linear systems for detection (example will be shown in section 4.2.4).

His constraint manager enables designers to generate geometric over-constraints iteratively (k⊖). When a geometric over-constraint is detected (l⊕), the constraint manager provides three alternatives, where users can select an appropriate one satisfying his/her needs. Finally, as the modeling is in equations (i⊕), his method is applicable to geometries of both free-form and Euler (g⊕⊖), linear and non-linear constraints (h⊕⊖) and 3D and 2D (j⊕⊖).

Results of all the evaluation results are summarized in the table 2.7.

Criteria set		Detection level			Decomposition			System modeling				Results generation	
Criteria	def	type	redundant conflicting	spanning group	over-constrained components	rigid subsystems	singular configuration	geometries	constraints	modeling	dimension	way of detection	debugging
Methods		a	b	c	d	e	f	g	h	i	j	k	l
Reduction	1	?	⊙	?	⊕	⊖	?	⊖	⊕	⊖	⊖	⊙	⊙
Dense	5	?	⊙	?	⊕	⊙	?	⊖	⊕	⊖	⊕⊖	⊙	⊙
Over-rigid	6	?	⊙	?	⊕	⊙	?	⊖	⊕	⊖	⊕⊖	⊙	⊙
MWM	7	⊖	⊙	⊙	⊕	⊖	?	⊖	⊕	⊖	⊕	⊕	⊕
D-M	8	⊖	⊙	⊙	⊕	⊖	⊖	⊕⊖	⊕⊖	⊕	⊕⊖	⊕	?
G-J	10	⊕	⊕	?	⊙	⊙	⊙	⊕⊖	⊖	⊕	⊕⊖	⊕	⊖
LU	10	⊕	⊕	?	⊙	⊙	⊙	⊕⊖	⊖	⊕	⊕⊖	⊕	⊖
QR	10	⊕	⊕	?	⊙	⊙	⊙	⊕⊖	⊖	⊕	⊕⊖	⊕	⊖
Grobner basis	10	⊕	⊕	⊖	⊙	⊙	⊖	⊕⊖	⊕	⊕	⊕⊖	⊖	⊖
Wu-Ritt	10	⊕	⊕	⊖	⊖	⊖	⊖	⊕⊖	⊕	⊕	⊕⊖	⊖	⊖
Perturbation	10	⊕	⊖	⊖	⊙	⊙	⊙	⊕⊖	⊕⊖	⊕	⊕⊖	⊕	⊖
NPM	10	⊕	⊖	⊖	⊖	⊖	⊖	⊕⊖	⊕⊖	⊕	⊕⊖	⊕	⊖
WCM	10	⊕	⊖	⊖	⊖	⊖	⊖	⊕⊖	⊕⊖	⊕	⊕⊖	⊕	⊖
WCM Extention	10	⊕	⊖	⊖	?	⊖	⊖	⊕⊖	⊕⊖	⊕	⊕⊖	⊖	?
hybrid-Serrano	10	⊕⊖	⊕	⊕	⊖	⊖	?	⊕⊖	⊕⊖	⊕	⊕⊖	⊖	⊕

Table 2.7: Evaluation of the methods

2.5 Benchmark and analysis of use cases

The previously introduced detection methods have been classified and summarized in table 2.7 which clearly shows their theoretical capabilities. The practical capabilities are studied in this section on different use cases.

2.5.1 Linear use case : Deformation of B-Spline curves

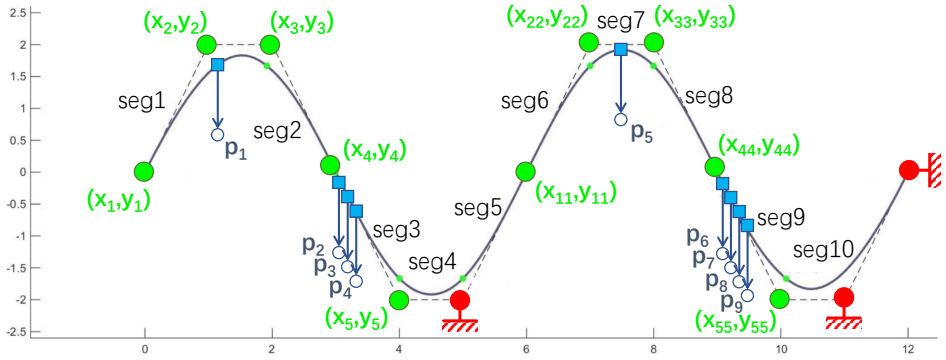


Figure 2.31: Constraints specification for the deformation of a B-Spline curve.

Problem description The first use case corresponds to the deformation of a B-spline curve of degree 3 defined by 13 control points and a knot sequence $U = \{u_0, u_0, u_0, u_0, u_1, \cdots, u_9, u_{10}, u_{10}, u_{10}, u_{10}\}$ with $u_0 = 0$ and $u_{10} = 1$. They are the values of knots, which are knowns. This knot sequence highlights 10 segments $[u_i, u_{i+1}]$ for which it will be important to understand whether they are under-, well- or over-constrained. Figure 2.31 shows the B-Spline curve with 10 of the control points (green circles) free to move and the remaining ones fixed (red triangles). The deformation is driven by 9 position constraints ($p1 - p9$). Finally, the deformation problem is defined by 10 couples of variables (xj, yj) which are the coordinates of pj and 18 equations ek with $k \in \{1..18\}$. Free control points and position constraints are presented in Table 2.8 together with the corresponding variables and equations.

control points	d_0^0	d_1^0	d_2^0	d_3^0	d_4^0	d_6^0	d_7^0	d_8^0	d_9^0	d_{10}^0
variables	$(x0, y0)$	$(x1, y1)$	$(x2, y2)$	$(x3, y3)$	$(x4, y4)$	$(x6, y6)$	$(x7, y7)$	$(x8, y8)$	$(x9, y9)$	$(x10, y10)$
constraints	$p1$	$p2$	$p3$	$p4$	$p5$	$p6$	$p7$	$p8$	$p9$	
equations	$(e1, e2)$	$(e3, e4)$	$(e5, e6)$	$(e7, e8)$	$(e9, e10)$	$(e11, e12)$	$(e13, e14)$	$(e15, e16)$	$(e17, e18)$	

Table 2.8: Typology of variables, constraints and equations involved in the use case of Figure 2.31.

Structural analysis : DoF counting As shown in figure 2.31, the curve is composed of 10 local segments according to the 10 intervals of the knots vector. Results of a manual DoFs analysis of each segment are presented in table 2.9. The curve is globally under-constrained but contains locally under-/well-/over-constrained subparts. However, DoF-based counting does not accurately reflect the non-solvability of this system.

	Seg1	Seg2	Seg3	Seg4	Seg5	Seg6	Seg7	Seg8	Seg9	Seg10
Affected by	$\{d_0^0 \dots d_3^0\}$	$\{d_1^0 \dots d_4^0\}$	$\{d_2^0 \dots d_5^0\}$	$\{d_3^0 \dots d_6^0\}$	$\{d_4^0 \dots d_7^0\}$	$\{d_5^0 \dots d_8^0\}$	$\{d_6^0 \dots d_9^0\}$	$\{d_7^0 \dots d_{10}^0\}$	$\{d_8^0 \dots d_{11}^0\}$	$\{d_9^0 \dots d_{12}^0\}$
DoFs	8	8	6	6	6	6	8	8	6	4
DoCs	2	0	6	0	0	0	2	0	8	0
Status	Under	Under	Well	Under	Under	Under	Under	Under	Over	Under

Table 2.9: DoF counting to define the status (under-, well-, over-constrained) of each local segment.

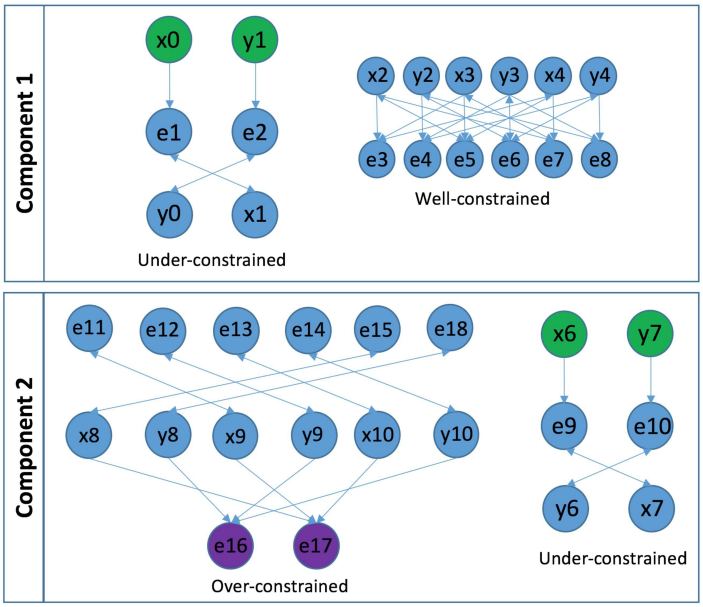


Figure 2.32: Results of applying BFS and D-M decomposition.

Structural analysis : BFS and D-M decomposition BFS allows to split the configuration into two local components which correspond to the upper and lower part of figure 2.32. D-M decomposition is then applied on each part. As illustrated on Figure 2.32, component 1 is decomposed into structurally under- and well-constrained subparts while component 2 is decomposed into structurally under- and over-constrained subparts.

More precisely, the result in Figure 2.32 shows that constraints $p1$ in Seg1 and $p5$ in Seg7 are structurally under-constrained, constraints $\{p6, p7, p8, p9\}$

in Seg9 are structurally over-constrained and the remaining constraints $\{p2, p3, p4\}$ in Seg3 are structurally well-constrained. The result matches well with able 2.9. Moreover, since equations $\{e16, e17\}$ are unmatched, constraints $\{p8, p9\}$ are structural over-constrained. Structural analysis using D-M decomposition thus gives the same results as the manual DoF analysis.

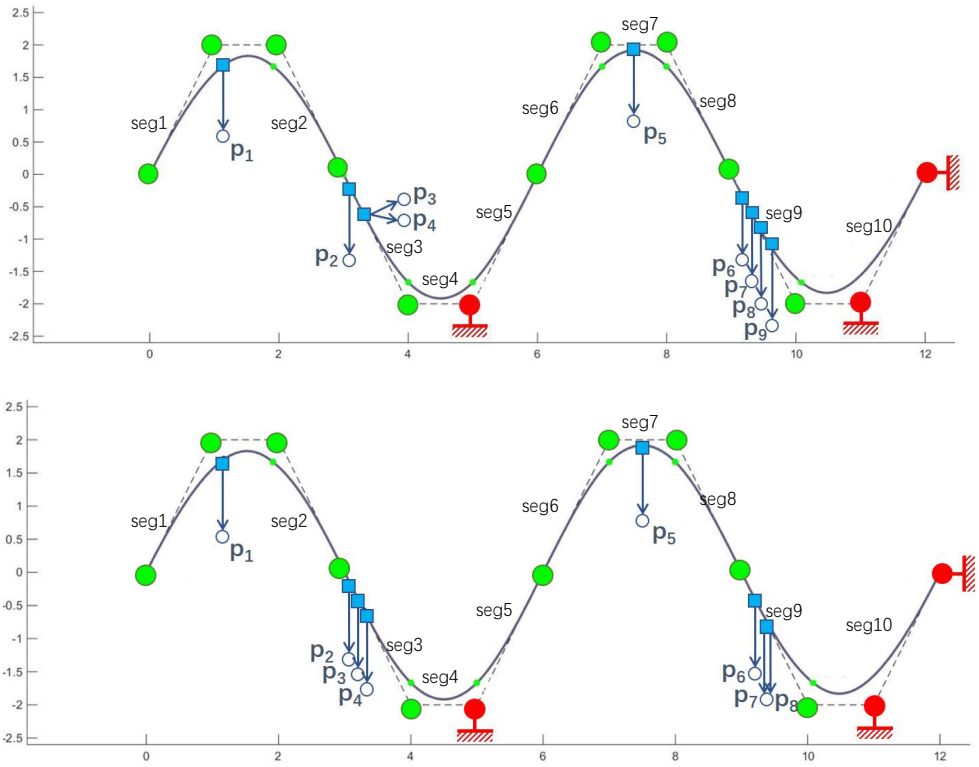


Figure 2.33: Curve 2 (Upper) with conflicting constraints $\{p3, p4\}$ and Curve 3 (Lower) with redundant constraints $\{p8, p9\}$.

However, D-M decomposition would fail when detecting numerical over-constraints. New examples are then introduced to illustrate such configurations. Figure 2.33 shows two new configurations obtained by modifying the previously defined constraints. For Curve 2, a point of the curve is assigned to two different positions ($p3 \neq p4$) thus creating a conflict. For Curve 3, a point of the curve is assigned to two similar constraints ($p8 = p9$) thus creating a redundancy.

As illustrated on figure 2.34, D-M decomposition gives the same results for the two configurations. Thus, the subtle difference between the two configurations cannot be distinguished, which confirms the need for further analyzing the system using numerical methods.

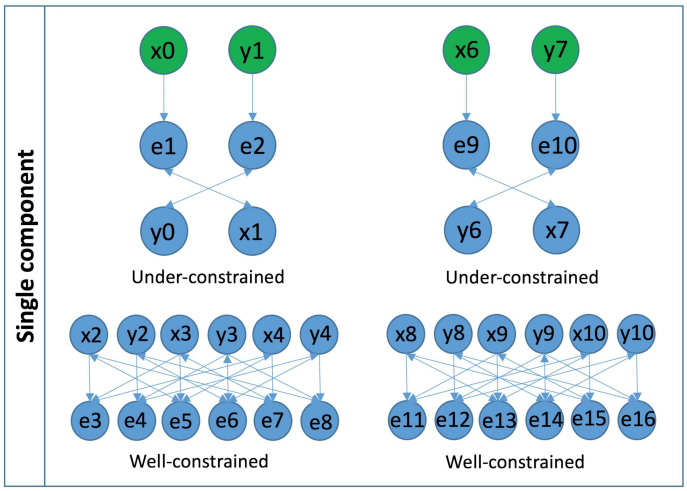


Figure 2.34: Result of D-M analysis for Curve 2 and Curve 3.

Figure 2.35: G-J analysis of Curve 2 (Upper) and G-J analysis of Curve 3 (Lower) with r the rank and n the number of equations.

Numerical analysis Since the constraints of the two configurations are linear, both G-J and QR can be used. Both methods identify equations $\{e7, e8\}$, respectively equations $\{e15, e16\}$, as the numerical over-constraints of Curve 2, respectively of Curve 3. However, in terms of fine detection, only G-J further identifies the conflicting and redundant equations. For Curve 2 (upper part of figure 2.35), equation $e7$ is conflicting (last term of the row

is not null) and $e8$ is redundant (last term of the row is null). Similarly, for Curve 3 (lower part of figure 2.35), equations $e15$ and $e16$ are redundant (zero values in the last column). Thus, as equations $\{e7, e8\}$ come from the same geometric constraint $p4$, respectively $\{e15, e16\}$ come from constraint $p8$, they cannot be split in two and the whole constraint $p4$ is to be considered as conflicting, respectively $p8$ considered as redundant.

	Structural analysis		Coarse detection		Fine detection
	BFS	D-M	G-J Elimination	QR Factorization	G-J Elimination
Curve 2	2 local parts	structural under- and well-constrained subparts for each local part	$e7$ and $e8$ over-constraints	$e7$ and $e8$ over-constraints	$e7$ conflicting $e8$ redundant
Curve 3	2 local parts	structural under- and well-constrained subparts for each local part	$e15$ and $e16$ over-constraints	$e15$ and $e16$ over-constraints	$e15$ and $e16$ redundant

Table 2.10: Testing results of Curve 2 and Curve 3

The result of the whole detection process is shown in table 2.10. For free-form linear constraints system, G-J is capable of conducting coarse and fine detection but QR is applicable for coarse detection only.

2.5.2 Non-linear use case : Double Banana

Problem description This section presents a non-linear use case with the Double Banana geometry (figure 2.36). The variables, constraints and the parameters of the Double Banana configuration are exactly the same as the one in the work of Moinet et al (Moinet, Mandil, and Serre, 2014). The difference is that their modeling is based on coordinate-free system while here it is Cartesian-based.

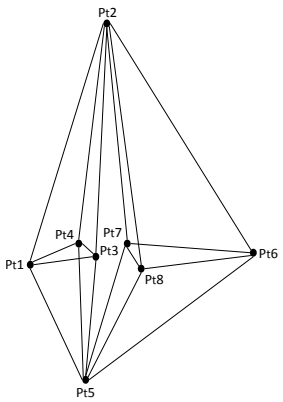


Figure 2.36: Initial geometry of the Double-Banana (Moinet, Mandil, and Serre, 2014).

As it is shown in figure 2.36, the Double Banana is defined by 8 points whose 3D coordinates are unknowns, thus there are 24 variables whose values are to be found. The Double Banana is constrained with 18 non-linear equations imposing distances between couples of points as detailed in figure 2.37.

#!VERTICES 8
Point: Pt1 (0.0000000000 ; 0.0000000000 ; 0.0000000000)
Point: Pt2 (-75.0000000000 ; 0.0000000000 ; 0.0000000000)
Point: Pt3 (-10.2666666667 ; 23.8871420550 ; -0.0000000000)
Point: Pt4 (-19.3801129874 ; -2.4055069381 ; 19.5356521540)
Point: Pt5 (22.9570528818 ; 19.9141617064 ; 37.1537331437)
Point: Pt6 (-34.3090195745 ; 59.7807966350 ; 15.7003332989)
Point: Pt7 (-33.9051208042 ; 17.3142000308 ; 40.1426130337)
Point: Pt8 (-19.4098849778 ; 29.6808580743 ; 25.5496727141)

#!EDGES 18
Edge: A1 (-Pt2;Pt1)
Edge: A2 (-Pt2;Pt3)
Edge: A3 (-Pt2;Pt4)
Edge: A4 (-Pt5;Pt1)
Edge: A5 (-Pt5;Pt3)
Edge: A6 (-Pt5;Pt4)
Edge: A7 (-Pt5;Pt6)
Edge: A8 (-Pt5;Pt7)
Edge: A9 (-Pt5;Pt8)
Edge: A10 (-Pt2;Pt8)
Edge: A11 (-Pt2;Pt7)
Edge: A12 (-Pt2;Pt6)
Edge: A13 (-Pt6;Pt7)
Edge: A14 (-Pt8;Pt6)
Edge: A15 (-Pt7;Pt8)
Edge: A16 (-Pt3;Pt4)
Edge: A17 (-Pt1;Pt3)
Edge: A18 (-Pt4;Pt1)

#!LENGTH SPECIFICATIONS 18
Length: l1 (A1) = 75
Length: l2 (A2) = 69
Length: l3 (A3) = 59
Length: l4 (A4) = 48
Length: l5 (A5) = 50
Length: l6 (A6) = 51
Length: l7 (A7) = 73
Length: l8 (A8) = 57
Length: l9 (A9) = 45
Length: l10 (A10) = 68
Length: l11 (A11) = 60
Length: l12 (A12) = 74
Length: l13 (A13) = 49
Length: l14 (A14) = 35
Length: l15 (A15) = 24
Length: l16 (A16) = 34
Length: l17 (A17) = 26
Length: l18 (A18) = 32

Figure 2.37: Parameters of the Double Banana configuration (Moinet, Mandil, and Serre, 2014).

Structural analysis: BFS and D-M decomposition As in the previous example, structural analysis methods are tested first. Results show that the whole system is only one connected component that is structurally under-constrained. However, we will see in the following that a conflicting constraint is therefore undetected.

Numerical analysis The results for coarse and fine detection are shown respectively in table 2.11 and table 2.12. Since two different sorting rows strategies are used, two different numerical over-constraints are detected ($e14$ for G-J and $e9$ for QR) and none of them is the same with the one found by Moinet et al. ($e18$). At the geometric level, those identified equations correspond to lengths imposed between points of the Double Banana. Thus, two aspects can be discussed when analyzing the results.

First, it is interesting to know how far are the lengths $\ell 14(A14)$, $\ell 9(A9)$ and $\ell 18(A18)$ from their initial specifications. This can be achieved while

Method	Over-constraint	*dev*	*cond*
G-J	$e14$	0.84/35	18.28
QR	$e9$	0.53/45	16.97
Moinet	$e18$	4.38/32	73.33

Table 2.11: Coarse detection using witness configuration on the Double Banana.

computing the deviation between the lengths that should be reached (i.e. imposed lengths as detailed in figure 2.37) and the lengths obtained by releasing the corresponding over-constraint, then solving and evaluating the new lengths. The column *dev* of table 2.11 gathers together the deviations when considering the identified over-constraints. For example, the identified over-constraint $e14$ is associated to the length $\ell 14(A14)$ initially set up to 35. If this equation is removed, the system can be solved and we obtained $\ell 14(A14) = 35.84$ which gives a relative deviation of $0.84/35 = 0.024$. The deviations obtained using G-J or QR are quite similar and rather small (i.e. about 10 times smaller) when compared to the deviation $4.38/32 \simeq 0.137$ obtained when removing $e18$ identified by the algorithm of Moinet et al (Moinet, Mandil, and Serre, 2014).

Then, it is interesting to know the degree of dependencies between constraints of the final system. This can be evaluated computing the condition number of the Jacobian matrix at the solution point of the new system generated after removing the over-constraints (i.e. $\ell 14(A14)$, $\ell 9(A9)$ or $\ell 18(A18)$). The results are shown in the column *cond* of the table 2.11. The dependencies between remaining constraints are lower for G-J and QR than for the method of Moinet et al.

Method	over-constraint	type	time (s)
Increm. solving G-J	$e14$	conflicting	174
Increm. solving QR	$e9$	conflicting	172
Grobner basis G-J	$e14$	?	?
Grobner basis QR	$e9$	?	?
Optimization G-J	$e14$	conflicting	633
Optimization QR	$e9$	conflicting	667
Moinet	$e18$	conflicting	–

Table 2.12: Fine detection using witness configuration on the Double Banana.

In terms of fine detection methods, table 2.12 shows that equations $e14$ and $e9$ are found to be conflicting by both incremental solving and optimization methods. The former method outperforms the latter because of less computational time and a ? is put in the table. However, Grobner-Basis does not converge in this amount of time. It is consistent with the discussion in (Latham and Middleditch, 1996) that Grobner-Basis is applicable for small systems with less than 10 non-linear algebraic equations. Finally,

the method by Moinet et al. also identifies $e18$ as a conflicting constraint but their paper does not address the time issue which can therefore not be compared.

Testing different witness configurations In the work of Moinet et al., the Double Banana configuration is linearized and the conflicting constraint $e18$ is detected from the linearized system (Moinet, Mandil, and Serre, 2014). Their linearization is on the witness configuration generated from the initial sketch. Here, this section extends their work by linearizing the system at a witness not generated from the initial sketch but from random configurations. The idea is to better understand whether the selection of a witness configuration affects the detection and identification of conflicting constraints.

Witness	1	2	3	4	5	6	7	8	9	10	11	12
G-J	$e14$	$e1$	$e6$	$e18$	$e4$	$e8$	$e16$	$e11$	$e13$	$e15$	$e12$	$e5$
Q-R	$e14$	$e1$	$e6$	$e18$	$e4$	$e8$	$e16$	$e11$	$e13$	$e15$	$e12$	$e5$
conflicting	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No	No	Yes	Yes
dev	0.84/35	3.3/75	-1.73/51	4.38/32	4.58/48	–	2.96/34	–	–	–	-2.45/74	-3.17/50

Table 2.13: Linearization at random witness to identify conflicting equations.

Table 2.13 gathers together the results of this analysis. It shows that there are cases where conflicting constraints are wrongly detected. here, witness configurations 6, 8, 9 and 10 do not allow a proper identification of conflicting constraints. For example, let us consider the witness configuration 9 illustrated on figure 2.38 and for which the coordinates of the points are given in table 2.14. This configuration is generic (Michelucci et al., 2006) because the rank of the matrix remains constant (and equal to 17 in the present case) even after a random perturbation on all the coordinates (figure 2.38 and table 2.14).

	node	1	2	3	4	5	6	7	8
witness 9	X	29	51	26	54	30	41	34	13
	Y	26	51	1	21	14	17	18	49
	Z	36	48	30	29	61	7	5	46
witness 9’	X	30	51	27	54	29	40	34	13
	Y	27	52	1	21	13	18	18	48
	Z	35	49	31	30	61	8	5	46

Table 2.14: Coordinates of witness configurations 9 and 9’ where the later is obtained by randomly inserting a perturbation in the coordinates of the former.

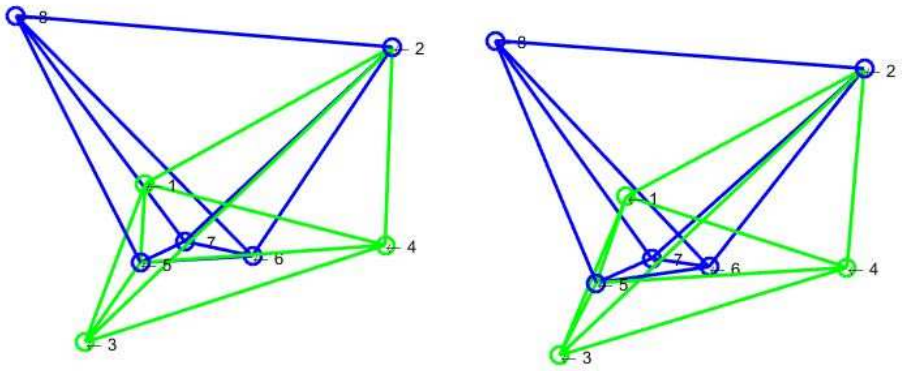


Figure 2.38: Left: Witness configuration 9 which does not allow a proper identification of the conflicting equations Right: Witness configuration 9' obtained from witness 9 with a random perturbation of amplitude $\epsilon = 1$.

Linear analysis identifies $e13$ as a conflicting constraint. We remove $e13$ and solve the remaining system using the solver based on Levenberg–Marquardt algorithm in MATLAB toolbox. The remaining system and the corresponding initial sketch where the solver starts to solve are shown in figure 2.39. The system is the same with the original one except that edge constraint $A13$ are removed. However, in this case, the solution process does not converge.

#!VERTICES 8
Point: Pt1 (0.0000000000 ; 0.0000000000 ; 0.0000000000)
Point: Pt2 (-75.0000000000 ; 0.0000000000 ; 0.0000000000)
Point: Pt3 (-10.2666666667 ; 23.8871420550 ; -0.0000000000)
Point: Pt4 (-19.3801129874 ; -2.4055069381 ; 19.5356521540)
Point: Pt5 (22.9570528818 ; 19.9141617064 ; 37.1537331437)
Point: Pt6 (-34.3090195745 ; 59.7807966350 ; 15.7003332989)
Point: Pt7 (-33.9051208042 ; 17.3142000308 ; 40.1426130337)
Point: Pt8 (-19.4098849778 ; 29.6808580743 ; 25.5496727141)

#!EDGES 18
Edge: A1 (-Pt2;Pt1)
Edge: A2 (-Pt2;Pt3)
Edge: A3 (-Pt2;Pt4)
Edge: A4 (-Pt5;Pt1)
Edge: A5 (-Pt5;Pt3)
Edge: A6 (-Pt5;Pt4)
Edge: A7 (-Pt5;Pt6)
Edge: A8 (-Pt5;Pt7)
Edge: A9 (-Pt5;Pt8)
Edge: A10 (-Pt2;Pt8)
Edge: A11 (-Pt2;Pt7)
Edge: A12 (-Pt2;Pt6)
Edge: A13 (-Pt6;Pt7)
Edge: A14 (-Pt8;Pt6)
Edge: A15 (-Pt7;Pt8)
Edge: A16 (-Pt3;Pt4)
Edge: A17 (-Pt1;Pt3)
Edge: A18 (-Pt4;Pt1)

#!LENGTH SPECIFICATIONS 18
Length: l1 (A1) = 75
Length: l2 (A2) = 69
Length: l3 (A3) = 59
Length: l4 (A4) = 48
Length: l5 (A5) = 50
Length: l6 (A6) = 51
Length: l7 (A7) = 73
Length: l8 (A8) = 57
Length: l9 (A9) = 45
Length: l10 (A10) = 68
Length: l11 (A11) = 60
Length: l12 (A12) = 74
Length: l13 (A13) = 49
Length: l14 (A14) = 35
Length: l15 (A15) = 24
Length: l16 (A16) = 34
Length: l17 (A17) = 26
Length: l18 (A18) = 32

Figure 2.39: System of removing e13

We can not reach a conclusion that the remaining system is non-solvable,

since the Levenberg-Marquardt algorithm converges to local minima rather than a global one. When applying an incomplete solver to solve system of equations, if the solving process converges, then the system is indeed solvable. Otherwise, complete solvers should be applied to know the solvability of a system, which will induce reliable results on determining redundant/conflicting status of an over-constraint.

2.6 Conclusion

This chapter has introduced several approaches for finding over-constrain ts and identifying conflicting as well as redundant equations in linear and non-linear geometric systems. Structural and numerical methods have been tested and analyzed with results on B-Spline curves and on the Double-Banana use cases. Several benefits have been reached. The definition of over-constraints with respect to free-form geometries is formalized. Local segments of free-form geometries can be found by BFS. Linear methods such as G-J and QR are capable of analyzing linear equations systems. These methods can also be used as part of the witness method to analyze non-linear equations systems. In this case, a particular attention has to be paid on the choice of the witness. In order to stay consistent with respect to the user-specified requirements, the discussion is first performed at the level of the equations but is then moved to the level of the geometry and design intent while considering the constraints encapsulating the different equations. As it is often the case in geometric constraint solving, large systems are first decomposed in subsystems which are then analyzed using numerical methods. Second, the treatment of the over-constraints could be extended while considering the variables as well as the objective function to be minimized as potential parameters to be found so as to get a solution closer to the design intent (The definition will be addressed formally in next chapter). Considering free-form geometries, a particular attention could be paid to the treatment of configurations where the knots of the knot sequences as well as the weights of the NURBS are considered as unknowns, and especially when considering free-form surfaces deformation. This will be further discussed in the next chapter.

Chapter 3

Detection and resolution in NURBS-based systems

In this chapter, we first introduce two detection frameworks (section 3.1) based on a combination of methods discussed in previous section. Then, we propose an original decision-support approach to address over-constrained geometric configurations. It focuses particularly on the detection and resolution of redundant and conflicting constraints as well as finding the corresponding spanning groups (section 3.4) when deforming NURBS patches. Based on a series of structural decompositions coupled with numerical analyses, the proposed approach handles both linear and non-linear constraints (section 3.2). Since the result of this detection process is not unique, several criteria are introduced to drive the designer in identifying which constraints should be removed to minimize the impact on his/her original design intent (section 3.3).

3.1 Detection framework: first scenarios

In real-life application, debugging geometric constraints systems can be done in two different ways. First, like within CAD modelers, designers are able to detect over-constraints interactively during the modeling process in 2D sketches (figure 1.9). In this case, the constraints are added incrementally. The other way of debugging is to analyze system of constraints that already exist. Here, it is assumed that all the constraints and associated equations have been predefined and that the analysis is to be performed on the whole system. Based on the work of previous section, we propose two detection frameworks to meet the two debugging ways: incremental detection and decremental detection. Both of them are based on a combination of structural and algebraic methods. These methods are listed in table 3.1 and table 3.2 respectively, and are discussed in the next subsection.

Level	Modeling	Method	Strong connected components
equation	bipartite graph	D-M	irreducible subsystems
geometry	bipartite graph	MWM	balanced sets

Table 3.1: Structural methods selected from table 2.7

	Linear Method	Non-linear Method
over-constraints	WCM	WCM
redundancies /conflicts	G-J/QR	Grobner basis /Incremental solving

Table 3.2: Algebraic methods selected from table 2.7

3.1.1 Incremental detection framework

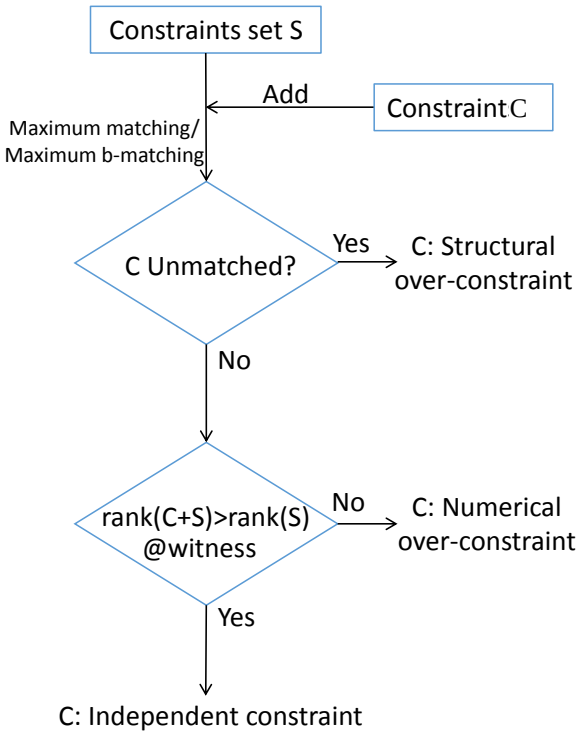


Figure 3.1: Incremental detection framework where constraints are added one by one

Here, we assume that the constraint C is to be added to a set of constraints S. This framework is to test if C is an over-constraint with respect to S. The first method we use is either D-M or MWM (table 3.1), which detects structural over-constraints through maximum matching or maximum b-matching. The method is applied to the new group $S + C$ after adding

C. If C is unmatched, then C is a structural over-constraint. Otherwise, we apply WCM method (table 3.2) to detect numerical over-constraints of $S + C$. If the rank of the new system $S + C$ is bigger than that of S at witness configurations, C is an independent constraint otherwise it is a numerical over-constraint. And whether it is redundant or conflicting can be checked using Grobner basis or Incremental solving method (figure 3.5). In this case, since the constraints has been added incrementally, the users can be informed directly if the newly inserted constraint has been detected as an over-constraint.

In the context of this PhD thesis, such a scenario is not necessarily the best one. As in this case, the detected over-constraints will not necessarily well affect the design intent, but rather the modeling process and its numerous modeling steps. However, one advantage is to be able to detect and treat the over-constraints as soon as they appear in the modeling process.

3.1.2 Decremental detection framework

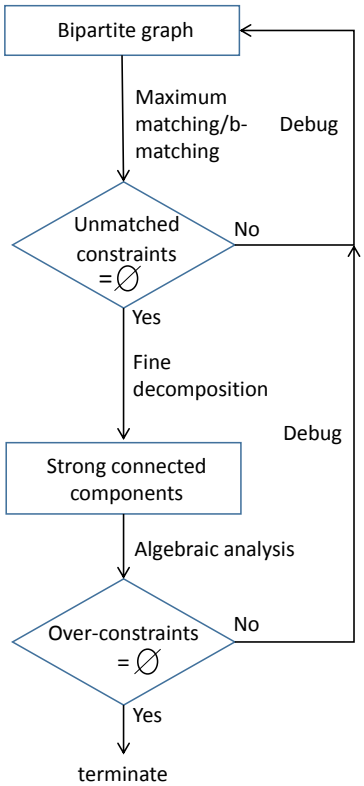


Figure 3.2: Decremental detection framework

Decremental detection analyzes a set of existing constraints. The constraints set and its associated equations set are initially represented with a bipartite graph. Structural over-constraints will be identified using either D-M or MWM (table 3.1) if there exists unmatched constraints after maximum matching (or b-matching). They will be removed and then the system is updated. If there is no unmatched constraints, strongly connected components (irreducible subsystems of D-M or balanced sets of MWM) are generated by fine decomposition of the system. Then, algebraic methods are used to detect numerical over-constraints inside each component. Since strongly connected components linked with solving order is actually a DAG structure, components corresponding to the source vertices are usually analyzed first. Once an over-constraint is found, it is removed. After that, the system is updated as well as the corresponding bipartite graph is rebuilt. The detection process finishes when no more numerical over-constraints are found.

In contrary to the previous scenario, the advantage here is that the decision on what to regarding the detected over-constraints can be performed on the entire system, thus better considering the design intent. At the opposite, if the system to checked is rather large, it can be more difficult to take decisions at the end, than to take decision during the modeling process.

3.1.3 Necessity of combining decomposition and algebraic methods

Drawbacks of algebraic analysis The previous sections have illustrated algebraic approaches as a mean to process system of equations and include:

- Symbolic methods for determining polynomial ideal membership in algebraically closed fields such as the Grobner basis and Wu-Ritt methods capable of distinguishing redundant and conflicting equations.

- Numerical methods analyzing Jacobian matrix of a system to find the rank deficient rows, such G-J, QR, NPM, and WCM: G-J and QR enable to distinguish linear redundant and conflicting equations while NPM and WCM can only identify over-constrained equations.

These direct algebraic solvers deal with general systems of polynomial equations, i.e. they do not exploit geometric domain knowledge. Symbolic methods have at least exponential time complexity and they are slow in practice as well. In addition, algebraic methods do not take into account design considerations and thus cannot assist in the conceptual design process.

Benefits of system decomposition Generally, decomposition exploits the local support property of NURBS geometry, which allows for knowing the distribution of local segments as well as the constrained status of each local parts. If some parts are locally over-constrained, users will treat these parts directly without globally inserting unnecessary control points. More specially, coarse decomposition returned by some methods enables a user for debugging purpose (e.g., identification of over-/under-constrained components), which corresponds to the view the user has of its system. For this reason, it is generally desirable to respect a coarse decomposition induced by a high-level user's manipulation of entities (e.g., mechanical components). Sitharam et al. proposed to adapt graph-based recursive assembly methods to this requirement (Sitharam, Peters, and Zhou, 2004). Moreover, decomposition methods are appealing for the drastic gain in efficiency they offer. For example, symbolic methods cannot be applied directly to large systems due to high computational cost but are applicable to the small subsystems after system decomposition.

Therefore, an algorithm should use geometric domain knowledge to develop a plan for locating local parts of a configuration, decomposing a local part into subsystem as small as possible so that algebraic methods could be applied recursively. Also, it is desirable if the algorithm provides coarse decomposition allowing for debugging purpose directly.

3.1.4 A Decomposition-Detection plan

The first requirement of a Decomposition-Detection (D-D) plan is that it should be able to find local segments of free-form configurations. Secondly, a D-D plan should decompose a constraint system into small subsystems and analyze these subsystems using algebraic methods. Since the time cost of over-constraints detection is proportional (at least polynomial) to the size of a system, the second requirement is that the small subsystems should be as small as possible so that algebraic methods can analyze them as fast as possible. If there is no over-constraints of a subsystem, it should be solved and the solution should be substituted into the entire system resulting in a simpler system. As it is shown in figure 3.3, a D-D plan initially decompose a system S into $\{S_1, \cdots, S_i, \cdots, S_n\}$ local segments using for instance the local support property of NURBS curves and surfaces. Then, for each local segment S_i, the D-D plan proceeds by iteratively applying the following steps at each iteration j:

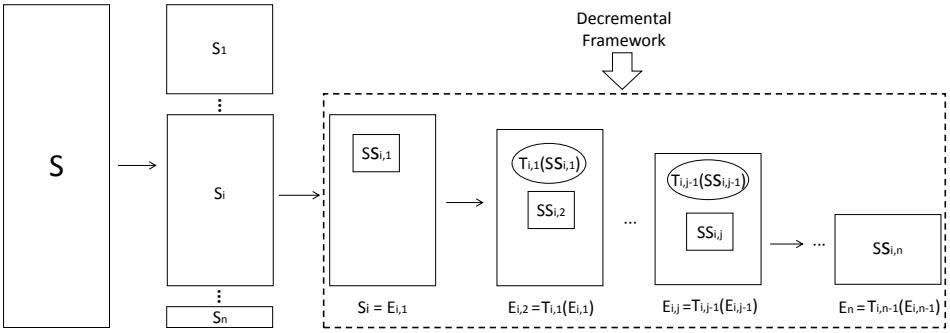


Figure 3.3: A Decomposition-Detection plan. S_i is the ith component after decomposition; S_{ij} is the component S_i at jth step when analyzing; E_{ij} refers to the entire system when analyzing S_{ij}.

1. Find the small subsystem $SS_{i,j}$ of the current local part S_i. Since the small subsystems are linked with solving sequence, the ones that are the source of the sequence should be chosen first ($SS_{i,1}$).

2. Detect numerical over-constraints in $SS_{i,j}$ using algebraic methods of Section 2.4.2. Users can either remove or modify them once they are detected. Otherwise, solve $SS_{i,j}$ directly using algebraic solver.

3. Replace $SS_{i,j-1}$ by an abstraction or simplification $T_{i,j-1}(SS_{i,j-1})$ thereby replacing the entire system $E_{i,j-1}$ by a simplification $E_{i,j} = T_{i,j-1}(E_{i,j-1})$. The simplification can be the removal/modification of the over-constraints or solving $SS_{i,j-1}$ and substituting the solution to $E_{i,j-1}$. The latter operation can potentially generate over-constraints since the solution of $SS_{i,j-1}$ may cause some equations of $E_{i,j-1}$ satisfied or unsatisfied (Section 3.3).

The decremental framework can be adapted and incorporated into a D-D plan to analyze S_i. In this way, S_i is initially represented with a bipartite graph. $SS_{i,j}$ corresponds to the strongly connected component of jth iteration, which is to be analyzed by algebraic method ($T_{i,j}(SS_{i,j})$). The analysis results could then be used to simplify $E_{i,j}$ through $T_{i,j}(E_{i,j})$. As a result, $E_{i,j}$ is updated to $E_{i,j+1}$. More details of the proposed decomposition plan will be given in Section 3.2

3.2 A generic approach coupling structural decompositions and numerical analyses

This section describes our approach for detecting and treating redundant and conflicting geometric constraints. The main idea is to decompose the system of equations into smaller blocks that can be analyzed iteratively using dedicated numerical methods. The overall framework and algorithm are first introduced before detailing the different steps involved (Hu, Kleiner, and Pernot, 2017).

3.2.1 Overall detection framework

The overall framework has been modeled in Figure 3.4. It is based on three nested loops: the structural decomposition into connected components (CC); the structural decomposition of a CC into its subparts ($G1, G2$, $G3$) and its corresponding DAG of stronglyly connected components (SCC); the iterative numerical analysis of these SCC. Pseudo-codes for the main procedures are provided in Section 3.2.3

Loop among connected components

The system of equations (SE) is initially represented by a graph structure G, where nodes correspond to variables and edges to equations. The structure is first decomposed into n connected components $\{CC_1, \cdots, CC_n\}$ using Breadth First Search (BFS) (Leiserson and Schardl, 2010). Such a decomposition is made possible thanks to the local support property of NURBS or simply when using constraints which decouple what happens along the x, y and z directions of the reference frame (e.g. position or coincidence constraints). As a result, geometric over-constraints can be detected separately for each CC_i.

Loop among subparts obtained by D-M decomposition

The D-M decomposition is used to structurally decompose CC_i into a maximum of three subparts: G_{i1} (over-constrained subpart), G_{i2} (well-constrained subpart) and G_{i3} (under-constrained subpart). Each subpart (if it exists) will be analyzed iteratively using the third nested loop explained below.

However, a single pass of the third loop on each G_{ij} is not sufficient. Indeed, any pass may lead to the removal of constraints, which modifies the

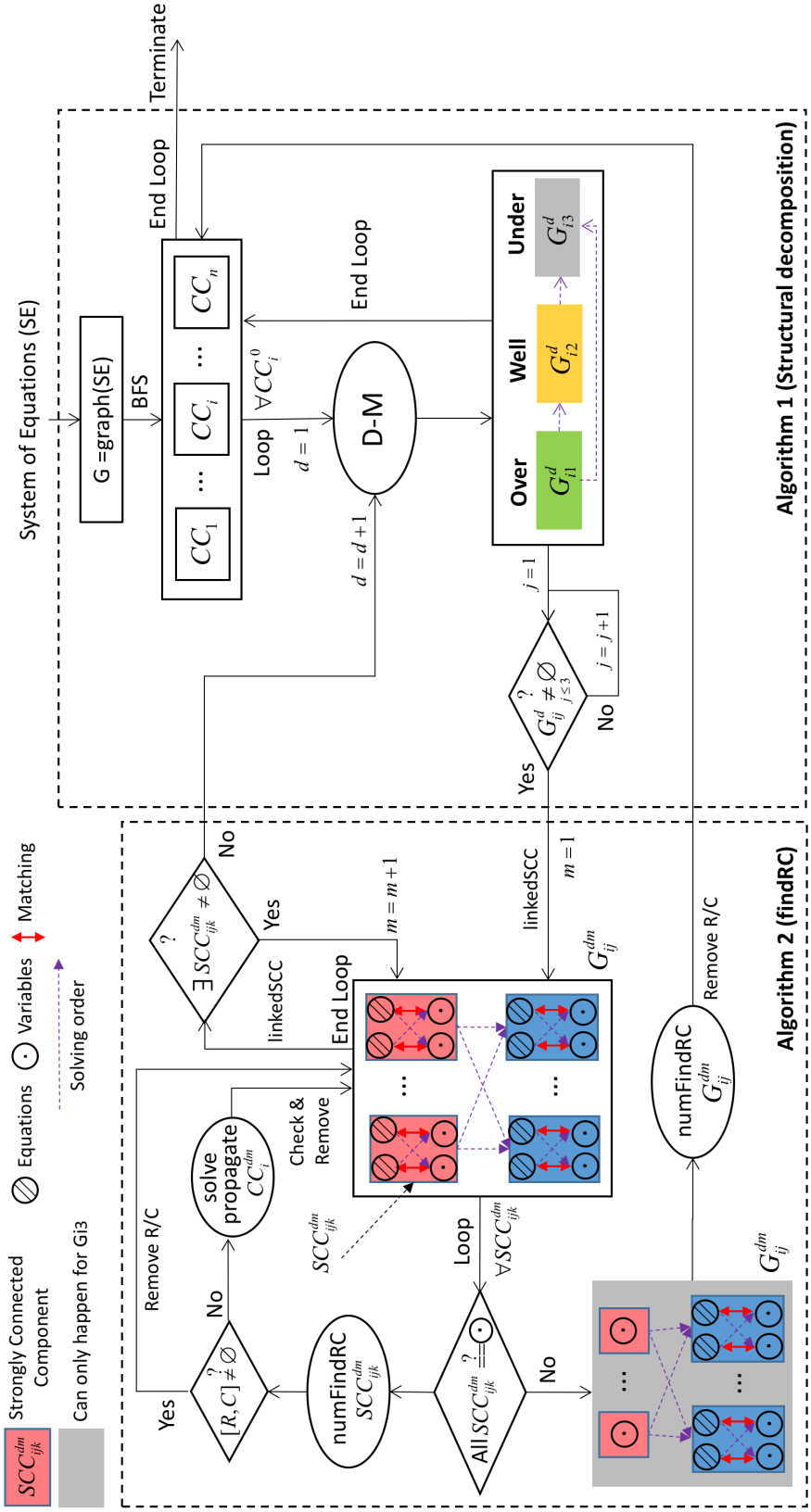


Figure 3.4: Overall framework composed of three nested loops defining the main structure of the detection algorithm

structure of the CC_i and thus requires to apply D-M decomposition again after the pass to obtain updated subparts. The superscript d is used to note that CC_i^d (resp. G_{ij}^d) refers to CC_i (resp. G_{ij}) after its d^{th} D-M decomposition. Although the number of passes required is unknown in advance, it is guaranteed that the process will converge to a state where only one subpart G_{i3} is left. In other words, constraints will be either removed or moved to the third subpart along the process.

Loop among stronglyly connected components

In addition to the subparts, D-M decomposition also provides a DAG for each CC_i^d. Nodes of this DAG are stronglyly connected components SCC_{ijk}^d. Edges of this DAG (purple arrows in the figure 3.4) denote solving dependencies between the SCC_{ijk}^d and may cross subparts G_{ij}^d boundaries. In the following, linkedSCC(G_{ij}^d) refers to the operation that obtains (the subpart of) this DAG from the d^{th} D-M decomposition of CC_{ij}^d that corresponds to the given subpart.

The third loop consists in trying to iteratively (in the DAG-dependencies induced order) find numerical over-constraints in each SCC_{ijk}^d or, when it is solvable, propagate its solution to other blocks. Since blocks are strongly connected, there is only one potential solution to each block unless it contains only variables, and this latter case can only be encountered in a third subpart G_{i3}^d. The process works only the top-level of the DAG (red blocks in the figure) because these blocks equations do not use variables from other blocks. Moreover, the other level of the DAG (blue blocks in the figure) use variables from other blocks, which contains the same number of variables and equations.

For each red block, and as shown in the top-left part of the figure 3.4, an appropriate numerical method (numFindRC in the figure and the pseudo-code) tries to find redundant (R) or conflicting (C) constraints. These over-constraints are then removed from the currently analyzed connected component CC_{ij}^d. If the block is solvable, its (unique) solution is propagated to dependent blocks, which may lead to additional redundant or conflicting constraints being detected and removed from CC_{ij}^d. Once all red blocks have been analyzed, this part of the DAG (potentially turning blue blocks into red ones) is recomputed until all blocks are analyzed. However, there is no need to recompute the D-M decomposition on the whole CC_{ij}^d and it is sufficient to recompute it only the maximum matching for the current subpart by calling linkedSCC again. The superscript m is used to note that G_{ij}^{dm} (resp. SCC_{ijk}^{dm}) refers to G_{ij}^d (resp. SCC_{ijk}^d) after its m^{th} matching. Although

the number of passes required is unknown in advance, it is guaranteed that the process will converge to a state where there are either no blocks left, or these blocks only contain variables (and that is only possible for the third subpart G^d_{i3}) according to our testing experiences . In other words, constraints and variables are removed until we obtain an under-constrained system with multiple solutions, meaning no more propagation is possible. In that last step, as shown in the bottom-left part of the figure, the remaining system is analyzed for numerical conflicts and proceeds with the next connected component.

3.2.2 Strongly connected components analysis

This section discusses the techniques used to analyze the stronglyly connected components SCC^{dm}_{ijk}. This corresponds to numFindRC function of Algorithm 2 (section 3.2.3) used to find redundant (R) and conflicting (C) constraints of a component if they exist. Otherwise the component is solved and solutions are propagated to the whole system. Here, QR is used for linear systems.

For non-linear systems, methods can be of three types. As discussed in section 2.5.2, symbolic methods like Grobner basis is limited to high computational cost when analyzing large systems of equations. Sometimes, the solving process cannot coverage (table 2.12). Numerical method like WCM, however, enables to detect numerical over-constraints of non-linear systems in polynomial time (Michelucci and Foufou, 2006a) but cannot further distinguish redundant and conflicting constraints. To find them, WCM should be used together with other methods that are able to distinguish redundant and conflicting constraints. In this section, we show how these methods are combined to detect redundant and conflicting constraints of a system. The different combinations are illustrated in figure 3.5 and discussed in next paragraphs.

Let us define a constraints system with a set of polynomial equations:

$$\begin{cases} f_1(x_1, x_2, \cdots, x_n) = 0 \\ f_2(x_1, x_2, \cdots, x_n) = 0 \\ \vdots \\ f_r(x_1, x_2, \cdots, x_n) = 0 \\ f_{r+1}(x_1, x_2, \cdots, x_n) = 0 \\ \vdots \\ f_m(x_1, x_2, \cdots, x_n) = 0 \end{cases} \tag{3.1}$$

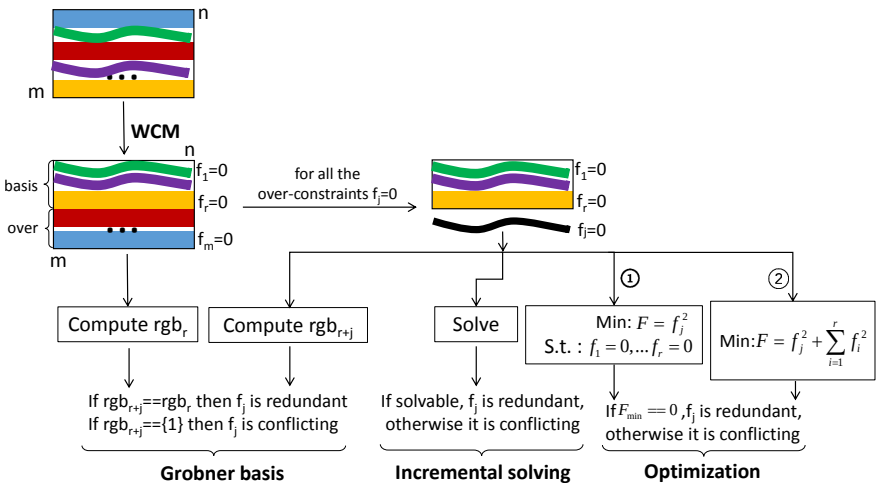


Figure 3.5: Combing WCM with Grobner basis, Incremental solving or Optimization to detect redundant and conflicting constraints.

By using WCM, we find that equations 1 to r are the *basis constraints* and the equations $(r+1)$ to m are the *over-constraints* (figure 3.5). To distinguish redundant and conflicting constraints, further steps are needed. They are detailed in the following paragraphs.

WCM+GB (Grobner Basis)

The Grobner Basis method for distinguishing redundant and conflicting equations can be briefly illustrated as follows. For a set of polynomials $f_0, f_1, \cdots, f_s \in \mathbb{C}[x_1, \cdots, x_n]$. Assuming the reduced Grobner basis(rgb_0) of the ideal $\langle f_1, \cdots, f_s \rangle$ satisfies $rgb_0 \neq \{1\}$ and $rgb_0 \neq \{0\}$ with respect to any ordering. To know if $f_0 = 0$ is a conflicting, redundant or independent equation, we compute the new reduced Grobner basis of the ideal $\langle f_0, f_1, \cdots, f_s \rangle$, which is rgb_{new}. If $rgb_{new} = \{1\}$, $f_0 = 0$ is a conflicting equation; if $rgb_{new} \equiv rgb_{old}$, $f_0 = 0$ is a redundant equation; if $rgb_{old} \subset rgb_{new}$, $f_0 = 0$ is an independent equation (paragraph 2.4.2).

As for distinguishing redundant and conflicting constraints in the over-constraints set, we first compute the reduced Grobner basis (rgb_{basis}) of the basis constraints ($\{f_1, \cdots, f_r\}$); then add one equation ($f_j = 0, j \in \{r+1, \cdots, m\}$) of the over-constraints list to the basis constraints and compute the reduced Grobner basis of the new group (rgb_{new}). Of course, rgb_{basis} satisfies $rgb_{basis} \neq \{1\}$ and $rgb_{basis} \neq \{0\}$ since equations contained are all independent. Moreover, rgb_{new} satisfies either $rgb_{new} = \{1\}$ or $rgb_{new} = \{0\}$, indicating $f_j = 0$ as redundant or conflicting respectively. Following the same

way, equations of the over-constraints are tested one by one with respect to the basis constraints. As a result, redundant and conflicting constraints in a set of the over-constraints are distinguished.

WCM+IS (Incremental Solving)

The Incremental Solving method test redundant and conflicting constraints by directly testing the solvability of system of equations. The solvability of basis constraints $\{f_1 = 0, \cdots, f_r = 0\}$ are tested first. Indeed, it is solvable. Then, we incrementally insert an over-constraint $f_j = 0$, $j \in \{r+1, \cdots, m\}$ to the set of basis constraints forming a new group of equations $\{f_1 = 0, \cdots, f_r = 0, f_j = 0\}$. If the new group is solvable, then $f_j = 0$ is redundant, otherwise it is conflicting.

WCM+OP(Optimization)

In the work of Ge et al (Ge, Chou, and Gao, 1999), optimization method is used to address geometric constraint satisfaction problems. They proposed to convert the system of equations $F = \{f_1, f_2, \cdots, f_n\}$ into the sum of squares $\sigma = \sum_{i=1}^{n} f_i^2$ and find the minimal value of σ. If σ_{min} is not equal to 0, the system is inconsistent. The method is mainly used to decide whether a system is consistent or not. In other words, it can be used to know the existence of conflicting constraints. However, we extend the method to two approaches (*Optimization 1* and *Optimization 2*) to test the redundancy or conflicting of $f_j = 0$, $j \in \{r+1, \cdots, m\}$ with respect to basis constraints$\{f_1 = 0, \cdots, f_r = 0\}$.

- *Optimization 1* A constraint satisfaction problem is set by transforming the over-constraint $f_j = 0$ to an objective function $F = f_j^2(X)$.

$$\begin{aligned} &\underset{X}{minimize} \quad F = f_j^2(X) \\ &subject\ to \quad \{f_1 = 0, \cdots, f_r = 0\},\ i = 1, \ldots r. \end{aligned}$$

  If F_{min} is equal to 0, then $f_j = 0$ is redundant; otherwise, it is conflicting.

- *Optimization 2* The constraints set $\{f_1 = 0, \cdots, f_r = 0, f_j = 0\}$ is transformed into CSP:

$$\underset{X}{minimize} \quad F = f_j^2(X) + \sum_{i=1}^{r} f_i^2(X)$$

  If F_{min} is equal to 0, then $f_j = 0$ is redundant; otherwise, it is conflicting.

3.2.3 Pseudo-code

Algorithm 3 Structural decomposition

SE ← System of Equations
G ← Graph(SE)
$[CC_1, \cdots, CC_n]$ ←BFS(G)
for $i = 1$ to n **do**
  $[G^1_{i1}, G^1_{i2}, G^1_{i3}]$ ←DM(CC_i)
  $CC^1_i \leftarrow CC_i$
  for $j = 1$ to 3 **do**
    $d \leftarrow 1$
    continue←True
    while continue & $G^d_{ij} \neq \varnothing$ **do**
      [continue,CC^{d+1}_i] ← findRC(CC^d_i, G^d_{ij})
      $d \leftarrow d + 1$
      $[G^d_{i1}, G^d_{i2}, G^d_{i3}]$ ←DM(CC^d_i)
    end while
  end for
end for
return $[CC^d_1, \cdots, CC^d_n]$

Algorithm 4 findRC: Numerical analysis of G^d_{ij} subpart of CC^d_i

Require: CC^d_i and G^d_{ij}
Ensure: Boolean continue and updated CC^d_i
$[SCC^{d1}_{ij1}, \cdots, SCC^{d1}_{ijN}]$ ←linkedSCC(G^d_{ij})
$m \leftarrow 1$
$G^{d1}_{ij} \leftarrow G^d_{ij}$
while $[SCC^{dm}_{ij1}, \cdots, SCC^{dm}_{ijN}] \neq \varnothing$ **do**
  $l \leftarrow 0$
  for $k = 1$ to N **do**
    if onlyVariable(SCC^{dm}_{ijk}) **then**
      $l \leftarrow l + 1$
    else
      $[R, C]$ ← numFindRC(SCC^{dm}_{ijk})
      if $[R, C] == \varnothing$ **then**
        solution ← solve(SCC^{dm}_{ijk})
        propagate(solution,CC^d_i)
        $R \leftarrow$ checkRedundant(CC^d_i)
        $C \leftarrow$ checkConflicting(CC^d_i)
      end if
      $CC^d_i \leftarrow$ removeRCfromCC($CC^d_i, [R, C]$)
    end if
    if $l == N$ **then** ▷ all red blocks contain only variables
      $[R, C]$← numFindRC(CC^d_i)
      $CC^d_i \leftarrow$ removeRCfromCC($CC^d_i, [R, C]$)
      return False,CC^d_i
    end if
  end for
  $G^{d(m+1)}_{ij} \leftarrow$ update(CC^d_i)
  $m \leftarrow m + 1$
  $[SCC^{dm}_{ij1}, \cdots, SCC^{dm}_{ijN}]$ ←linkedSCC(G^{dm}_{ij})
end while
return True,CC^d_i

As it is shown in above tables, we provide the pseudo-code for the two main procedures of the approach, surrounded by dotted rectangles on figure 3.4.

3.2.4 Rough time complexity analysis

Assuming systems of equations have V number of variables and E number of equations. It is initially decomposed into N number of unconnected components. Then, for each component i, DM decomposition is applied M_i times to analyze the DAGs (figure 3.6). For one DAG j, we assume that there are E_{ij} number of equations and V_{ij} number of variables as well as K_{ij} number of red blocks. In these blocks, we assume that there are K_{ij-non} number of blocks containing non-linear equations and K_{ij-lin} number of blocks containing linear equations. Also, for one block K_k, we assume that there are m_{ijk} number of equations and n_{ijk} number of variables.

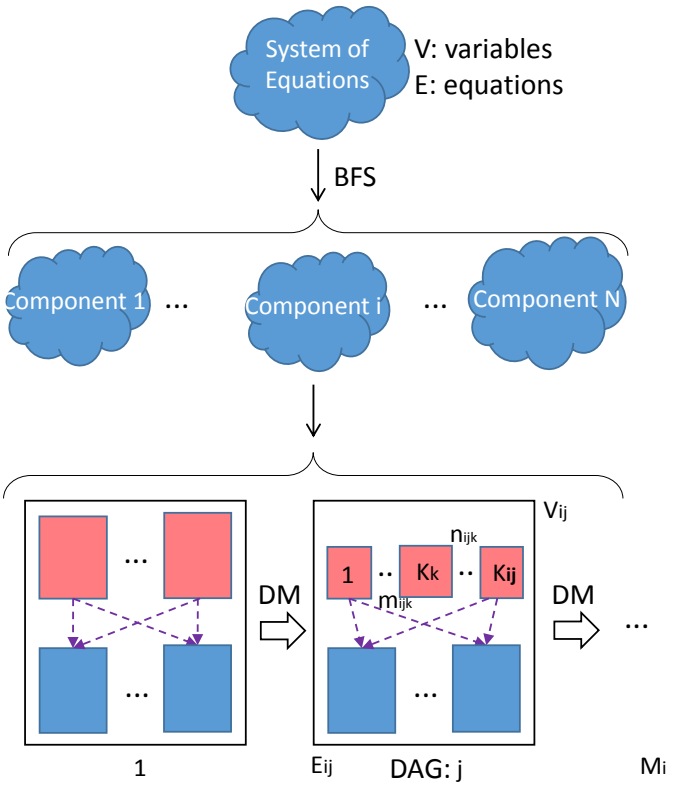


Figure 3.6: Decomposition for time complexity analysis.

Operation	BFS	D-M	Grobner basis	QR
Time complexity	$O(V+E)$	$O(E_{ij}V_{ij})+O(E_{ij})$	$2\left(d^2/2+d\right)^{2^{n_{ijk}-1}}$	$O(n_{ijk}^3)$

BFS: Operate on the whole system of equations
D-M: Operate on one of the DAGs (jth-DAG) (Pothen and Fan, 1990)
Grobner basis: Operate on one of the red blocks (kth red block). n_{ijk} is the number of variables, and d is the maximal total degree of the input polynomials (Dubé, 1990)

Table 3.3: Time complexity for individual operations

Time complexity of the methods used in the basic operations is summarized in table 3.3 As a result, by adding basic operations together, the total time complexity can be estimated as:

$$O(V+E)+\sum_{i=1}^{N}\sum_{j=1}^{M}\left[O(E_{ij}V_{ij})+O(E_{ij})+\sum_{k=1}^{K_{ij-non}}2(d^2/2+d)^{2^{n_{ijk}-1}}+\sum_{k=1}^{K_{ij-lin}}n_{ijk}^3\right] \tag{3.2}$$

The above time complexity estimates the upper bound computation time for detecting over-constraints that are detected by QR or Grobner basis method. However, for detecting over-constraints using Optimization or Incremental solving, the time complexity is hard to estimate because solving nonlinear equations is an iterative process, whose convergence speed is problem-dependent. Therefore, estimate the time complexity of the whole algorithm by considering only the number of equations and variables is not enough. Actually, time complexity estimation considering only number of equations and variables can be misleading in some cases. Empirical experiments on time complexity will be shown in section 4.4

3.3 Validation and evaluation of the solutions

Section 1.3 has introduced the multiple ways to model requirements within an optimization problem by specifying an unknown vector X, the constraints to be satisfied $F(X) = 0$ and the function $G(X)$ to minimize.

The approach described in this section allows for the identification of redundant and conflicting equations. Correctness is ensured since it consists of a fixed-point algorithm that only stops when the system is solvable. Additionally, any removed equation is guaranteed to be either conflicting or redundant with the remaining set. It has thus been shown that the set of equations $F(X) = 0$ can be decomposed in two subsets: $F_b(X) = 0$ containing the basis equations, and $F_o(X) = 0$ the over-constrained ones.

To stay close to the requirements the designer has in mind, the proposed approach then moves from the equations level to the constraints level. Thus,

the geometric constraints associated to the equations $F_o(X) = 0$ are analyzed and all the equations related to those constraints are gathered together in a new set of equations $\widetilde{F}_o(X) = 0$. Of course, the equations $F_o(X) = 0$ are included in the set of equations $\widetilde{F}_o(X) = 0$. Finally, the equations related to constraints which are neither conflicting nor redundant form the other set $\widetilde{F}_b(X) = 0$. This transformation allows working at the level of the constraints and not at the level of the equations. This is much more convenient for the end-user interested in working at the level of geometric requirements.

Since this decomposition is not unique, it gives rise to various potential final solutions (interactive decomposition has not been considered in this thesis). Therefore several criteria are now introduced to evaluate these solutions according to the initial design intent. To be able to characterize the quality of the obtained solutions, the set of user-specified parameters P is introduced. This set gathers together all the parameters the designer can introduce to define the constraints his/her shape has to satisfy. For example, the distance d imposed between two points of a NURBS surface is a parameter characterizing a part of the design intent. Then, the idea is to evaluate how much the solutions deviate from the initial design intent and notably in terms of the parameters P.

To do so, the optimization problem containing the basis constraints is solved :

$$\begin{cases} \widetilde{F}_b(X) = 0 \\ \min G(X) \end{cases} \tag{3.3}$$

and the solution X' is then used to evaluate the unsatisfied over-constraints $\widetilde{F}_o(X')$ as well as the real values P' of the user-specified parameters P. For example, if the user-specified distance d between the two patches cannot be met, then the real distance d' will be measured on the obtained solution. From this solution, it is then possible to evaluate three quality criteria:

- *Deviation in terms of parameters/constraints*: this criterion aims at measuring how far/close the real values P' of the parameters are from the user-specified parameters P. This criterion helps understanding if the design intent is preserved in terms of parameters and consequently in terms of constraints.

$$df = \frac{\sum_i |P'_i - P_i|}{\sum_i |P_i|} \tag{3.4}$$

- *Deviation in terms of function to minimize*: this criterion directly evaluates how much the objective function G has been minimized. Here,

the function is simply computed from the solution X' of the optimization problem. To preserve the design intent this value is to be minimized. Thus, it can be used to compare several solutions between them.

$$dg = G(X') \tag{3.5}$$

- *Degree of near-dependency*: rank deficiency of the Jacobian matrix at the witness clearly reveals the dependencies between constraints. However, for NURBS-based equation systems, the constraints can be independent but near to be dependent. In this case, the Jacobian matrix of $\widetilde{F_b}(X)$ at the solution point X' is ill-conditioned and the corresponding solution can be of bad quality. The third criterion thus evaluates the condition number (*cond*) of the Jacobian matrix as a measure of near-dependency (Kincaid and Cheney, 2002):

$$cond = cond(\mathbb{J}_{\widetilde{F_b}}(X')) \tag{3.6}$$

Finally, even if those criteria characterize the quality of the solution X' with respect to the design intent, they have not been combined in a unique indicator. Thus, the results of the next chapter will be evaluated by analyzing and comparing those three criteria for each solution.

3.4 Finding spanning groups

Among one loop of analyzing stronglyly connected components, numerical over-constraints can be found by either analyzing strongly connected components directly or indirectly generated during the process of propagating solutions. In this section, we first discuss how the spanning groups of these over-constraints can be found in one loop and then discuss how to link the spanning groups of different loops together so as to obtain the final spanning groups of the over-constraints.

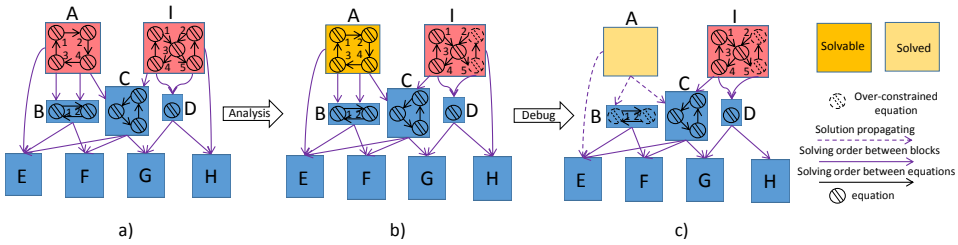


Figure 3.7: Detecting over-constraints in one loop: An example

3.4.1 In one loop

In this section, we assume that the DAG structure of one loop is shown in figure 3.7(a). Block A is solvable while block I contains over-constraints. After solving block A and propagating the solutions to the other blocks, all the over-constraints in block B are either satisfied or unsatisfied. For the former, spanning groups of over-constraints of block I are inside the block itself. For the later, however, spanning groups of over-constraints of block B are outside the block since these over-constraints are triggered by solutions of block A.

Inside block

For an over-constraint E_{oi} of a set of over-constraints E_o in a block, its spanning group E_{sg} is the set of basis constraints E_b of that block. Since in a strongly connected component every vertex is reachable from every other vertex, the over-constraint can be reached from all the basis constraints through the variables they share.

Outside block

In this case, the spanning group of an over-constraint gathers together the constraints of the block whose solution triggers the satisfactory or unsatisfactory of the over-constraint.

Taken figure 3.7 as an example. At step a), block A and I are analyzed. The result shows that block A is solvable while I contains numerical over-constraints which are $I.2$ and $I.5$. As discussed in previous section, the spanning group of the over-constraints are $\{I.1, I.3, I.4\}$. In step c), we solve the block A and propagate the solution to the DAG structure. As a result, constraints in block B $\{B.1, B.2\}$ are the over-constraints with constraints in block A as the spanning group $\{A.1, A.2, A.3, A.4\}$. They are summarized in table 3.4.

type	over-constraints	spanning groups
I	I.2,I.5	I.1,I.3,I.4
II	B.1,B.2	A.1,A.2,A.3,A.4

Table 3.4: Two types of over-constraints and spanning groups

Thus, there exist two types of over-constraints: over-constraints detected based on the basis constraints inside the same block (type I), or over-constraints detected based on the solutions from other blocks (type

II). For the former, they are detected by numerical methods discussed in section 3.2.2. For the latter, they are detected during the process of propagating solutions.

Finding the spanning group of type II

Spanning group of type I is easy to find, since the basis constraints are within the same block of an over-constraint. In this paragraph, we propose a method for finding the spanning group of type II based on a directed graph. We take figure 3.7 as an example. Since constraints of block B are fully fed by solutions from block A, the spanning group can be traced back through the feeding variables.

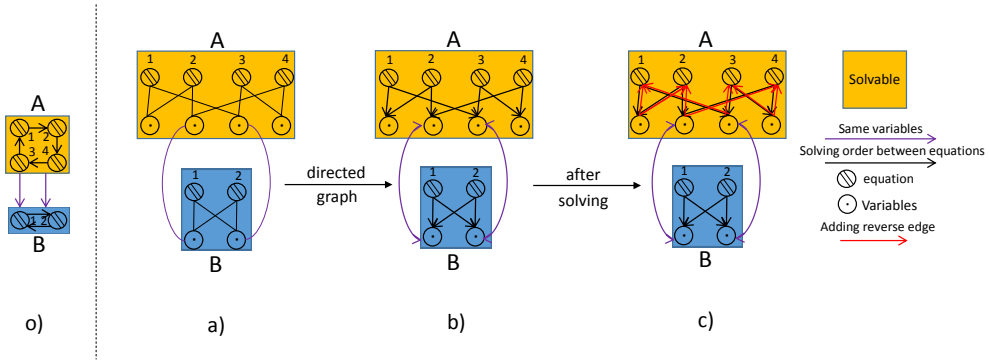


Figure 3.8: Finding spanning group of type II of figure 3.7.

Steps to find spanning groups are:

1. Equation graphs are generated according to the equations. Note that, here we assume that the variables within a block are shared by every two equations so that the block is a strongly connected component(figure 3.8 a).

2. Directed graphs are generated by transforming edges into directed edges where a direction is pointed from an equation to a variable. As it is shown in figure 3.8 b), we also use a bi-directed edge to link the feeding variables between the two blocks. These variables are exactly the same variables.

3. After solving, new directed edges are generated by pointing from variables to edges. As it is shown in step c) of the figure, they are the red edges in block A by simply reversing the edges of step b).

4. Starting from B.1 and B.2, the spanning groups are the equations reached along the directed edges, which are the set of equations $\{A.1, A.2, A.3, A.4\}$ in this case.

3.4.2 Linking loops together

The previous paragraph has shown how to find the spanning group of an over-constraint in one loop of D-M decomposition. However, as it is shown in figure 3.4, the DAG structure G_{ij}^{dm} is updated frequently when looping on analyzing SCC_{ijk}^{dm}s, which means that a constraint may go through many loops before it is finally detected as an over-constraint. For example, in figure 3.9, three groups of over-constraints are detected during the three loops of applying D-M decomposition (table 3.5).

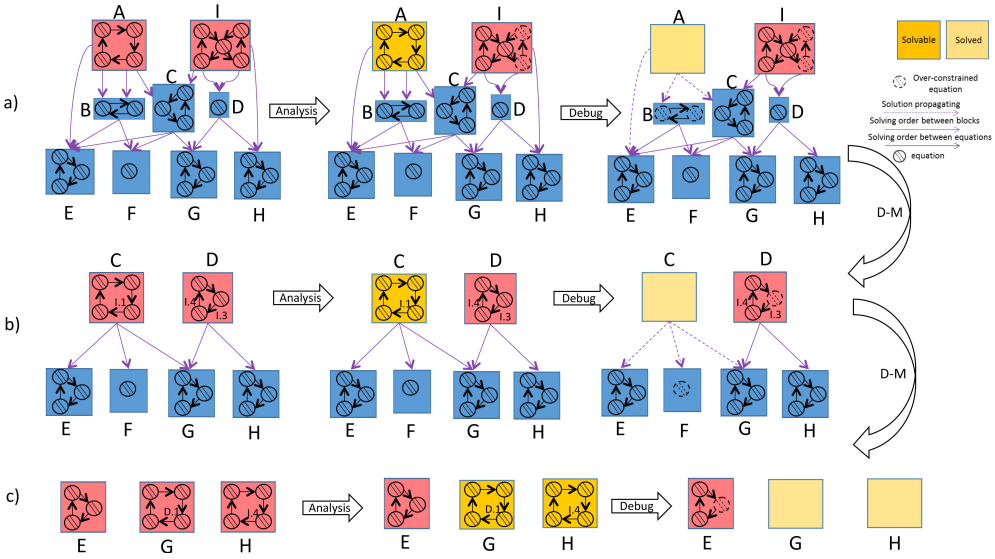


Figure 3.9: Detecting over-constraints along several D-M loops. a) First loop b) Second loop c) Third loop

Over-constraints	First loop	Second Loop	Third loop
type I	I.2,I.5	D.1	E.3
type II	B.1,B.2	F.1	

Table 3.5: Over-constraints detected in three loops of figure 3.9

The spanning group of the three over-constraints are summarized in the table 3.6 Spanning group of D.1 is $\{I.3, I.4\}$ which are directly from red block I. Spanning group of E.3 is more complicated, However. In loop c),

the spanning group is $\{E.1, E.2\}$. But in loop b), the solution of block C is propagated to the equations of block E, resulting in red block E in loop c). Therefore, equations in block E of loop c) is equivalent with the equations in block C and E of loop b). The spanning group of E.3 is extended to $\{E.1, E.2, C.1, C.2, C.3, I.1\}$. Similarly, in loop a), the solution of block A updates the structure of block C. By adding equations of block A, the final spanning group of $E.3$ is $\{E.1, E.2, C.1, C.2, C.3, I.1, A.1, A.2, A.3, A.4\}$. Comparing to E.3, F.1 is of type II and thus does not have spanning group inside block F. The final spanning group is a set of equations outside the block (table 3.6, third column).

Over-constraint	D.1	E.3	F.1
Spanning group	I.3,I.4	E.1,E.2,C.1,C.2,C.3,I.1,A.1,A.2,A.3,A.4	C.1,C.2,C.3,I.1,A.1,A.2,A.3,A.4

Table 3.6: Spanning groups of the over-constraint: D.1, E.3 and F.1

3.4.3 General case

In our algorithm, variables of a constraint can be substituted at different loops of D-M decomposition before it is finally detected as an over-constraint. For example, in figure 3.10, variables of constraint E.3 and F.1 are fed by solution from block C which has been fed by the solution from block A. More general case of constraint E.3 and F.1 are shown in figure 3.11 a) and b) respectively. Variables of an over-constraint or basis constraints can be substituted at different D-M loops before the final identification.

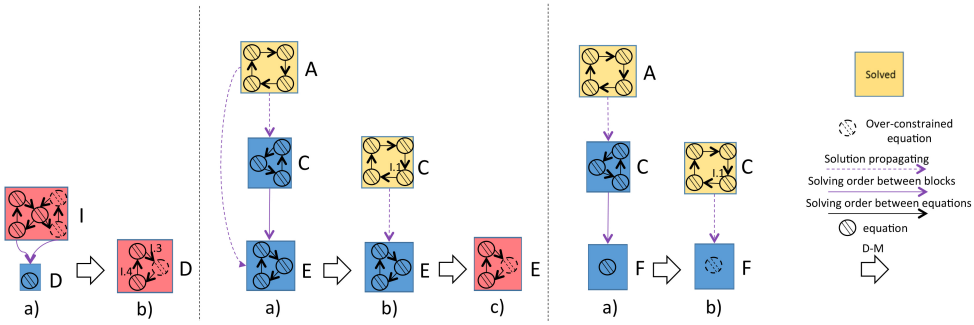


Figure 3.10: Loops of detecting the over-constraint: D.3 (left), E.3 (middle) and F.1 (right)

The final spanning group is a combination of the spanning group in each D-M loop. For case a) of figure 3.11, the spanning group of each loop is a set of equations of the yellow blocks whose solution contributes directly or indirectly to the substitution of the variables of the over-constraint. Note

that, we use E_i to represent a set of equations in a block. For example, in step n-1) of figure 3.12, solutions of equation set E6 are substituted directly to the over-constraint(type II) while solutions of equation set E4 and E5 are firstly propagated to equation set E2 and E3 respectively and then in step n), the solution of E2 and E3 are directly fed to the over-constraint. In the latter case, solutions of E4 and E5 are fed *indirectly* to the over-constraint in step n-1).

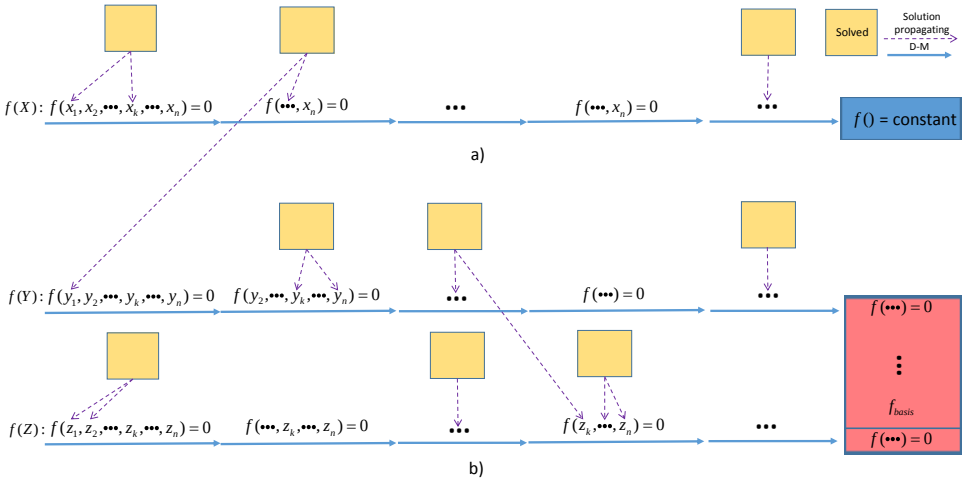


Figure 3.11: General feeding process of over-constraints

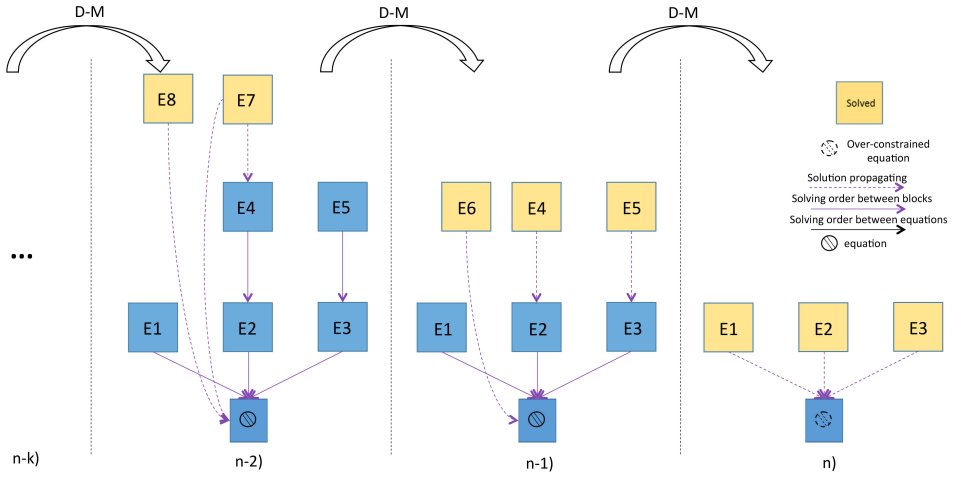


Figure 3.12: Finding spanning group during D-M decompositions

However, for an over-constraint of type I, the spanning group of each loop is a set of equations of the yellow blocks whose solution contributes di-

rectly or indirectly to the substitution of the variables of the over-constraint as well as the corresponding basis constraints. As it is shown in figure 3.13, in step n-1), the spanning group are E4 which spans the over-constraint; E5 and E6 which span the basis constraints.

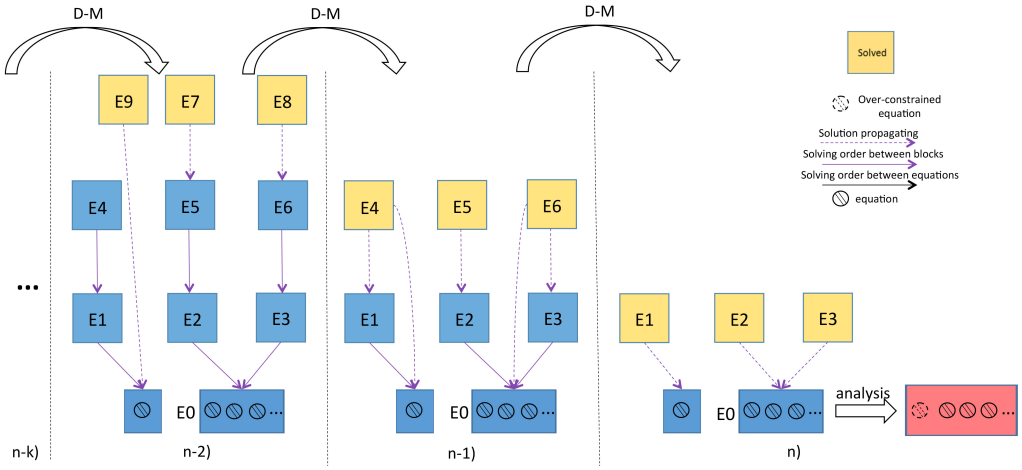


Figure 3.13: Finding spanning group during D-M decompositions

3.4.4 Pseudocode

This section provides the pseudo-code for finding the spanning groups. We modified the previous algorithms and generated the new ones.

Algorithm 5 Structural decomposition

SE ← System of Equations
$G \leftarrow$ Graph(SE)
$[CC_1, \cdots, CC_n] \leftarrow$BFS($G$)
for $i = 1$ to n **do**
  $[G^1_{i1}, G^1_{i2}, G^1_{i3}] \leftarrow$DM($CC_i$)
  $CC'_i \leftarrow$ directedGraph(CC_i) ▷ directions are from equations to variables
  $CC^1_i \leftarrow CC_i$
  $CC'^1_i \leftarrow CC'_i$
  for $j = 1$ to 3 **do**
    $d \leftarrow 1$
    continue←True
    while continue & $G^d_{ij} \neq \varnothing$ **do**
      [continue , CC^{d+1}_i, CC'^{d+1}_i]← findRCandSpanningGroup($CC^d_i, G^d_{ij}, CC'^d_i$)
      $d \leftarrow d + 1$
      $[G^d_{i1}, G^d_{i2}, G^d_{i3}] \leftarrow$DM($CC^d_i$)
    end while
  end for
end for
return $[CC^d_1, \cdots, CC^d_n]$

Algorithm 6 findRCandSpanningGroup: Find redundant/conflicting constraints and their spanning groups

Require: CC_i^d, G_{ij}^d and $CC_i'^d$

Ensure: Boolean continue, updated CC_i^d, $CC_i'^d$, $spanningGroupR$, $spanningGroupC$

1: $[SCC_{ij1}^{d1}, \cdots, SCC_{ijN}^{d1}] \leftarrow$linkedSCC($G_{ij}^d$)
2: $m \leftarrow 1$
3: $G_{ij}^{d1} \leftarrow G_{ij}^d$
4: **while** $[SCC_{ij1}^{dm}, \cdots, SCC_{ijN}^{dm}] \neq \varnothing$ **do**
5: $l \leftarrow 0$
6: **for** $k = 1$ to N **do**
7: **if** onlyVariable(SCC_{ijk}^{dm}) **then**
8: $l \leftarrow l + 1$
9: **else**
10: $[R, C] \leftarrow$ numFindRC(SCC_{ijk}^{dm})
11: **if** $[R, C] == \varnothing$ **then**
12: solution $\leftarrow$ solve(SCC_{ijk}^{dm})
13: $CC_i'^d \leftarrow reverseEdge(CC_i'^d, SCC_{ijk}^{dm})$ ▷ reverse the edges of SCC_{ijk}^{dm} in $CC_i'^d$: from variables to equations
14: propagate(solution,CC_i^d)
15: $R \leftarrow$ checkRedundant(CC_i^d)
16: $spanningGroupR \leftarrow descendantEquations(CC_i'^d, R)$ ▷ Find all the descendant equations starting from R in $CC_i'^d$.
17: $C \leftarrow$ checkConflicting(CC_i^d)
18: $spanningGroupC \leftarrow descendantEquations(CC_i'^d, C)$ ▷ Find all the descendant equations starting from C in $CC_i'^d$.
19: **else**
20: $spanningGroupR \leftarrow descendantEquations(CC_i'^d, R \cup SCC_{ijk}^{dm}.basis)$ ▷ Find all the descendant equations starting from R and the basis equations in SCC_{ijk}^{dm}.
21: $spanningGroupC \leftarrow descendantEquations(CC_i'^d, C \cup SCC_{ijk}^{dm}.basis)$ ▷ Find all the descendant equations starting from C and the basis equations in SCC_{ijk}^{dm}.
22: **end if**
23: $CC_i^d \leftarrow$ removeRCfromCC(CC_i^d, $[R, C]$)
24: **end if**
25: **if** $l == N$ **then** ▷ all red blocks contain only variables
26: $[R, C] \leftarrow$ numFindRC(CC_i^d)
27: $CC_i^d \leftarrow$ removeRCfromCC(CC_i^d, $[R, C]$)
28: **return** False, $CC_i^d, CC_i'^d$
29: **end if**
30: **end for**
31: $G_{ij}^{d(m+1)} \leftarrow$ update(CC_i^d)
32: $m \leftarrow m + 1$
33: $[SCC_{ij1}^{dm}, \cdots, SCC_{ijN}^{dm}] \leftarrow$linkedSCC($G_{ij}^{dm}$)
34: **end while**
35: **return** True, $CC_i^d, CC_i'^d$

One can notice that those algorithms do not really return the redundant and conflicting equations as they are removed during the detection process. Spanning groups are identified and the list of equations is saved outside the algorithm.

3.5 Conclusion

In this chapter an approach for finding all over-constraints in free-form geometric configurations has been introduced. It relies on a coupling between structural decompositions and numerical analysis. The approach has several benefits: it is able to distinguish between redundant and conflicting constraints; it is applicable on both linear and non-linear constraints; and it applies numerical methods on small sub-blocks of the original system, thus allowing to scale to some large configurations. Additionally, since the set of over-constraints of a system is not unique, it has been shown that our approach is able to provide different sets depending on the selected structural decomposition, and proposed criteria to compare them and assist the user in choosing the constraints he/she wants to remove. Even if the kernel of the algorithm works on equations and variables, the decision is taken by considering the geometric constraints specified by the designer at a high level. Finally, method on finding the spanning group of an over-constraint is proposed. For an over-constraint, there may exist many spanning groups. Our algorithm enables to find one of them based on a directed graph. In the next chapter, the detection and resolution processes as well as finding spanning groups are illustrated with results on both academic and industrial examples.

Chapter 4

Results and discussion

This chapter presents three configurations on which the proposed over-constraints detection and resolution technique has been tested. The first one concerns the academic Double-Banana testing case widely studied in the literature (section 4.1). It has been used to compare our solution to the ones generated by others. The two other examples are more industrial and concerns the shaping of a teapot (section 4.2) and glass (section 4.3) composed of several NURBS patches. The relationship between time complexity and size of a system, and the influence of specified tolerances on the detection results are discussed in section 4.4.

4.1 Double-Banana testing case

The variables X, the constraints $F(X) = 0$, and the parameters P of the Double-Banana testing case are exactly the same as the ones tested by Moinet et al. (Moinet, Mandil, and Serre, 2014). The configuration as well as the parameters are discussed in section 2.5.2.

4.1.1 Modeling

The system to be solved is composed of 18 equations (figure 4.1) and 24 variables. The variables are the coordinates (x_i, y_i, z_i) of 8 vertices (Pt_i) of the Double-Banana geometry. The incidence matrix of the equations is shown in table 4.1

e1: x1^2 - 2*x1*x2 + x2^2 + y1^2 - 2*y1*y2 + y2^2 + z1^2 - 2*z1*z2 + z2^2 - 5625 = 0
e2: x2^2 - 2*x2*x3 + x3^2 + y2^2 - 2*y2*y3 + y3^2 + z2^2 - 2*z2*z3 + z3^2 - 4761 = 0
e3: x2^2 - 2*x2*x4 + x4^2 + y2^2 - 2*y2*y4 + y4^2 + z2^2 - 2*z2*z4 + z4^2 - 3481 = 0
e4: x1^2 - 2*x1*x5 + x5^2 + y1^2 - 2*y1*y5 + y5^2 + z1^2 - 2*z1*z5 + z5^2 - 2304 = 0
e5: x3^2 - 2*x3*x5 + x5^2 + y3^2 - 2*y3*y5 + y5^2 + z3^2 - 2*z3*z5 + z5^2 - 2500 = 0
e6: x4^2 - 2*x4*x5 + x5^2 + y4^2 - 2*y4*y5 + y5^2 + z4^2 - 2*z4*z5 + z5^2 - 2601 = 0
e7: x5^2 - 2*x5*x6 + x6^2 + y5^2 - 2*y5*y6 + y6^2 + z5^2 - 2*z5*z6 + z6^2 - 5329 = 0
e8: x5^2 - 2*x5*x7 + x7^2 + y5^2 - 2*y5*y7 + y7^2 + z5^2 - 2*z5*z7 + z7^2 - 3249 = 0
e9: x5^2 - 2*x5*x8 + x8^2 + y5^2 - 2*y5*y8 + y8^2 + z5^2 - 2*z5*z8 + z8^2 - 2025 = 0
e10: x2^2 - 2*x2*x8 + x8^2 + y2^2 - 2*y2*y8 + y8^2 + z2^2 - 2*z2*z8 + z8^2 - 4624 = 0
e11: x2^2 - 2*x2*x7 + x7^2 + y2^2 - 2*y2*y7 + y7^2 + z2^2 - 2*z2*z7 + z7^2 - 3600 = 0
e12: x2^2 - 2*x2*x6 + x6^2 + y2^2 - 2*y2*y6 + y6^2 + z2^2 - 2*z2*z6 + z6^2 - 5476 = 0
e13: x6^2 - 2*x6*x7 + x7^2 + y6^2 - 2*y6*y7 + y7^2 + z6^2 - 2*z6*z7 + z7^2 - 2401 = 0
e14: x6^2 - 2*x6*x8 + x8^2 + y6^2 - 2*y6*y8 + y8^2 + z6^2 - 2*z6*z8 + z8^2 - 1225 = 0
e15: x7^2 - 2*x7*x8 + x8^2 + y7^2 - 2*y7*y8 + y8^2 + z7^2 - 2*z7*z8 + z8^2 - 576 = 0
e16: x3^2 - 2*x3*x4 + x4^2 + y3^2 - 2*y3*y4 + y4^2 + z3^2 - 2*z3*z4 + z4^2 - 1156 = 0
e17: x1^2 - 2*x1*x3 + x3^2 + y1^2 - 2*y1*y3 + y3^2 + z1^2 - 2*z1*z3 + z3^2 - 676 = 0
e18: x1^2 - 2*x1*x4 + x4^2 + y1^2 - 2*y1*y4 + y4^2 + z1^2 - 2*z1*z4 + z4^2 - 1024 = 0

Figure 4.1: System of equations of the Double Banana geometry

	x1	y1	z1	x2	y2	z2	x3	y3	z3	x4	y4	z4	x5	y5	z5	x6	y6	z6	x7	y7	z7	x8	y8	z8
e4	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
e1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1
e9	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1
e2	0	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e18	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
e10	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
e7	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0
e15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
e13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0
e12	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0
e8	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0
e16	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
e17	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e6	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
e3	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
e5	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
e11	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0

Table 4.1: Incidence matrix of the equations of the Double Banana geometry

4.1.2 Detection process

Incidence matrix describes the relationships between variables and equations. Adjacency matrix describes the relationships between variables, which can be obtained from incidence matrix. Based on the adjacency matrix, the constraint graph between variables is drawn in figure 4.2.

Finding local parts

The Breadth First Search (BFS) is used to find the unconnected components of the constraint graph. As revealed by BFS, the constraint graph contains only one component, meaning that there is only one local part in the whole system.

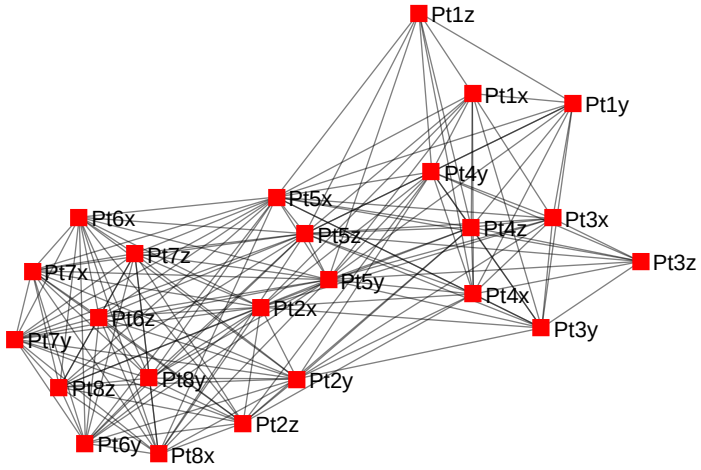


Figure 4.2: Constraint graph of the variables of the Double Banana geometry

Analysis of the local parts

	z4	z6	z7	x8	y8	z8	x1	y1	z1	x2	x3	y3	x4	y4	x5	y5	z5	y2	z2	x6	y6	x7	y7	z3
e17	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
e18	1	0	0	0	0	0	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
e4	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
e1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e2	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	0	0	0	0	0
e16	1	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1
e6	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0
e3	1	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	1	1	0	0	0	0	0
e9	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0
e7	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	1	0	0	0
e8	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	1	1	0
e10	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0
e12	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	1	0	0	0
e14	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0
e13	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
e15	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
e11	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	1	1	0
e5	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1	0	0	0	0	0	0	1

Table 4.2: Incidence matrix after D-M decomposition

Structural analysis For the local part, D-M decomposition is used for structural analysis. As it is shown in table 4.2, the whole part is structurally under-constrained. Maximum matching of the part is the diagonal (marked red) of the matrix. Strongly connected components and the solving sequence between them form a DAG structure which is shown in figure 4.3. Since all the red blocks are only variables, the whole part should be analyzed together. Thus, the process follows the bottom part of the algorithm of figure 3.4.

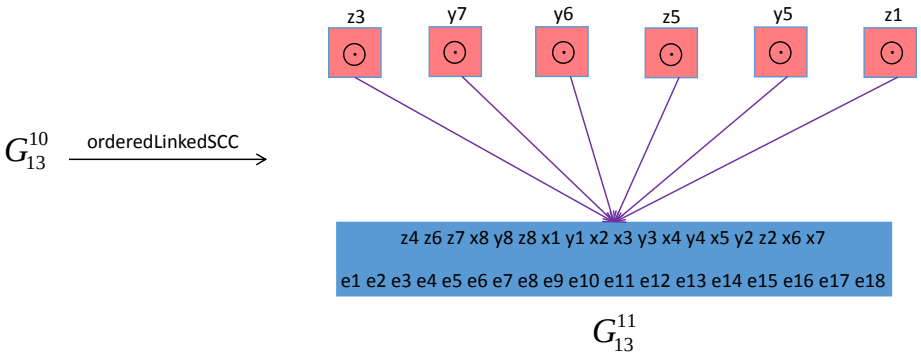


Figure 4.3: Evolution of G_{13}^{1}

Numerical analysis WCM analysis is used within our numFindRC function and an over-constraint is detected. More specifically, the equation e9 is here detected. Using our Incremental Solving approach, the equation is further characterized as a conflicting one (type I).

Component	Initial D-M			D-M	Over-constraints			
	Over	Well	Under	times	Redundant	Type	Conflicting	Type
1			√	1			$e9$	I

Table 4.3: Result of analyzing Double-Banana configuration

The equation $e9$ is therefore removed and the system is solved using the initial position of the nodes as initial values of the variables. Using the results, the equation $e9$ is then reevaluated and the associated parameter is compared to the user-specified value. In the present case, $e9$ is not satisfied since it is equal to 44.47 compared to the initial user-specified requirement of 45 (figure 2.37). Thus, the deviation from the design intent is $df = 0.53/45$. Our algorithm gives a solution that is much closer to the initial design intent than the algorithm of Moinet et al., and the remaining system is less ill-conditioned after removing the conflicting constraint (table 4.4). Actually, the algorithm of Moinet et al. identifies $e18$ as a conflicting

equation and its removal induces a deviation $df = 4.38/32$ from the initial design intent.

Method	Witness	Over-constraint	*df*	*cond*
Our	initial sketch	e9	0.53/45	16.97
Moinet et al.	initial sketch	e18	4.38/32	73.33

Table 4.4: Comparison between our algorithm and Moinet's approach on the Double-Banana testing case

It has been shown that our algorithm works well on this geometry. To see if it works well on the other configurations as well, we introduce the following industrial example: a 3D teapot, whose geometry type and constraints are much more complex.

4.2 Sketching a 3D teapot

In this example, we show how the proposed over-constraints detection and resolution approach can support the sketching of a 3D teapot (figure 4.4). The designer sketches the teapot following his/her design intent and the associated requirements.

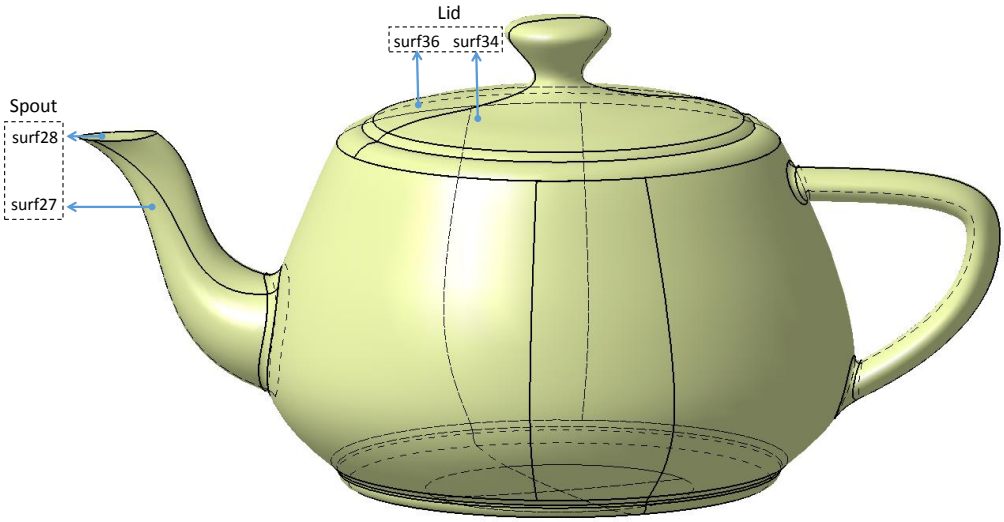


Figure 4.4: Initial sketch of a teapot

4.2.1 Modeling

Here, the objective is to modify the spout and lid of the teapot by specifying the following elements:

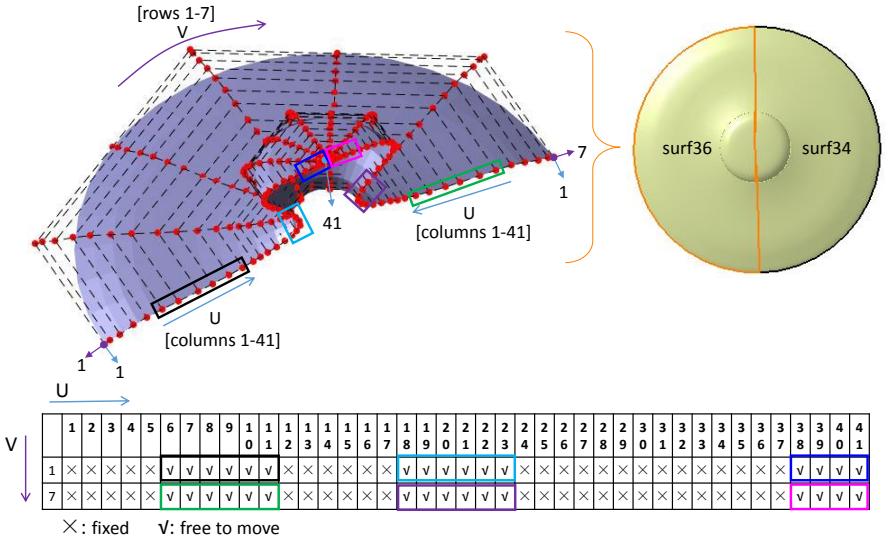


Figure 4.5: Free control points of the Lid

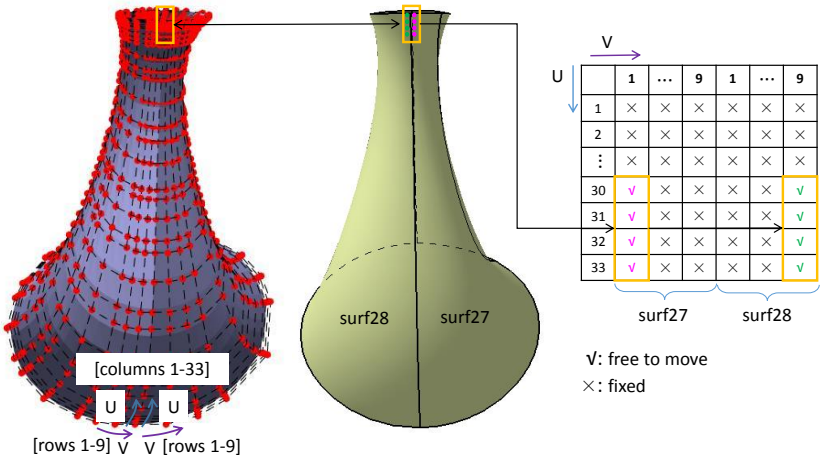


Figure 4.6: Free control points of the Spout

- *Variables*: As it is shown in figure 4.5, the lid is composed of two symmetric patches (surf36 and surf34). Each patch has a degree 6×3 and has a control polygon made of 41×7 control points. Similarly, the spout is composed of two symmetric patches (surf27 and surf28), each of which has a degree 6×6 and has a control polygon made of 33×9 control points (figure 4.6). Positions of free control points

with respected to the control polygon are indicated in the table of both figures and their coordinates are the variables of our optimization process. Since the objective is to modify the dimension of the teapot, the designer selects how many rows of control points are to be blocked and how many can move. For example, in figure 4.5, free control points of row 1 and 7 are selected to modify the shape of surf36. As a result, there are $2 \times 16 \times 3 = 96$ variables. Results will be illustrated with free control points of the same columns in rows [1,7] and rows [1,2,6,7] for both surf34 and surf36. For surf27 and surf28, free control points are in rows [1,9] and [1,2,8,9] along V direction as well as columns [30,31,32,33] along U direction (figure 4.6).

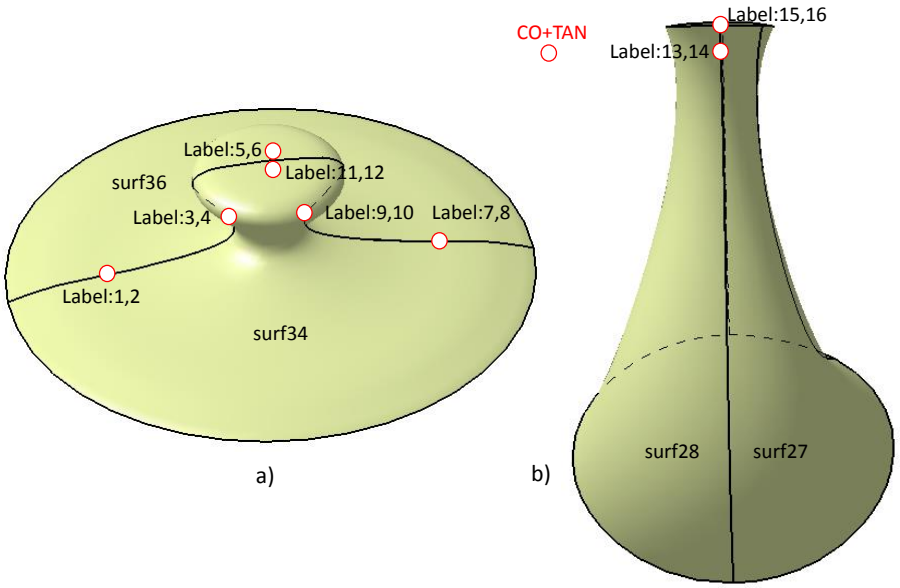


Figure 4.7: Coincidence and tangency constraints of a) the Lid, b) the Spout

- *Constraints*: Two types of constraints are used to specify how the shape of the 3D teapot has to evolve:

  - *Coincidence and tangency*: In figure 4.7, 8 coincidence and 8 tangency constraints are specified to maintain the continuity between surf34 and surf36, surf27 and surf28. They are labeled from 1 to 16 and they generate 8×3 linear and 8×3 non-linear equations labeled from 1 to 48 (table 4.5). The non-linearity comes from the use of the vector product to express the collinearity of normals.

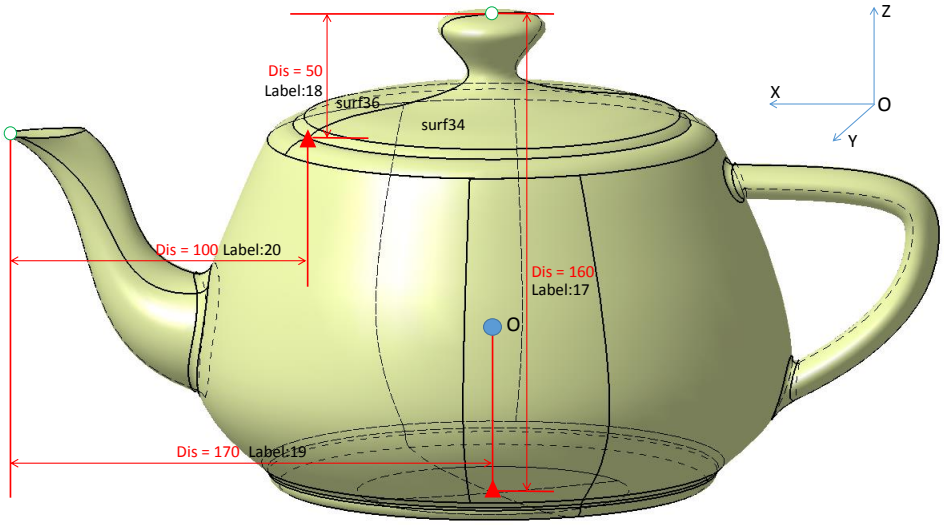


Figure 4.8: Distance constraints of teapot

– *Distance*: 2 distance constraints are defined between the lid and the bottom along the Z direction, and 2 distance constraints between the spout and the bottom along the X direction respectively (figure 4.8). They are labeled 17 and 18, 19 and 20. The corresponding equations are equations 49 and 50, 51 and 52 (table 4.5).

Constraint	Equations	Type	Component
1	1-3	linear	1
2	4-6	non-linear	1
3	7-9	linear	3
4	10-12	non-linear	3
5	13-15	linear	5
6	16-18	non-linear	5
7	19-21	linear	2
8	22-24	non-linear	2
9	25-27	linear	4
10	28-30	non-linear	4
11	31-33	linear	6
12	34-36	non-linear	6
13	37-39	linear	8
14	40-42	non-linear	8
15	43-45	linear	7
16	46-48	non-linear	7
17	49	linear	5
18	50	linear	5
19	51	linear	7
20	52	linear	7

Table 4.5: Typology of constraints and equations involved in the description of the 3D teapot sketching example

Overall, there are 20 geometric constraints generating 52 equations in the set $F(X) = 0$. Some of those constraints are added intentionally conflicting and it is the purpose of this section to try to see how our algorithm can detect and remove them without affecting too much the design intent at the level of the constraints.

4.2.2 Detection Process

Finding local parts

BFS decompose the whole system into 59 unconnected components with 51 of them containing only variables (figure 4.9). Each of the other 8 components contains equations and they are analyzed further by our algorithm.

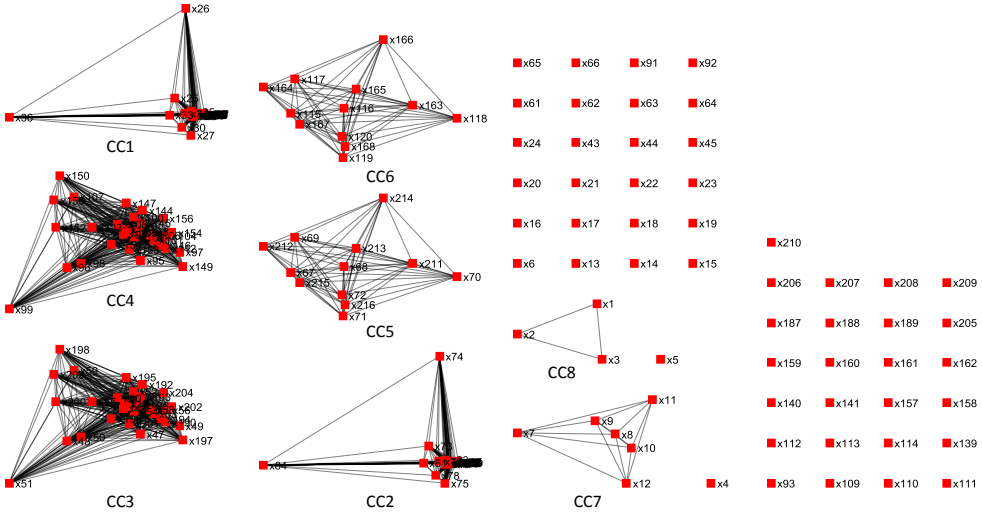


Figure 4.9: Constraint graph between variables

Analysis of the local parts

Analyzing component 1 (CC1) Incidence matrix between variables and equations is shown in table 4.6. The evolution of G_{11}^{10} is shown in figure 4.10. Since all the red blocks contains only variables, all the equations of CC1 are analyzed at one time. As a result, equation $e6$ is redundant (type I). The spanning group of $e6$ is $\{e1, e2, e3, e4, e5\}$.

Analyzing component 2 (CC2) Component CC2 is very similar to component CC1. That is, their equations are different only in the name of variables and equations. Following the similar analysis process of CC1, equation

	x25	x26	x27	x28	x29	x30	x31	x32	x33	x34	x35	x36	x37	x38	x39	x40	x41	x42	x169	x170	x171	x172	x173	x174	x175	x176	x177	x178	x179	x180	x181	x182	x183	x184	x185	x186
e1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
e2	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
e3	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1
e4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
e5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
e6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 4.6: Component 1 (CC1 6×36)

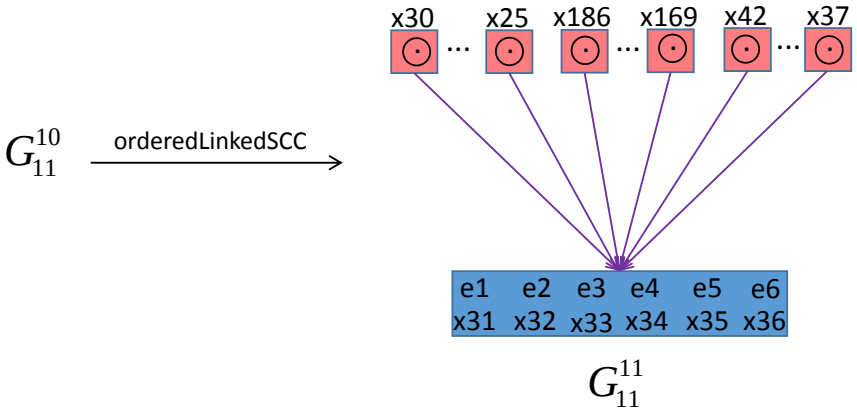


Figure 4.10: Evolution of G_{11}^{1}

$e24$ is redundant(type I). It is redundant with $\{e19, e20, e21, e22, e23\}$, which corresponds to the spanning group.

Analyzing component 3 (CC3) Component CC3 contains 6 equations with 30 variables. The red blocks of the DAG structure (figure 4.11) contain only variables and thus all the system of equations are analyzed together. As a result, equation $e10$ is redundant (type I). It is redundant with $\{e7, e8, e9, e11, e12\}$ which corresponds to the spanning group.

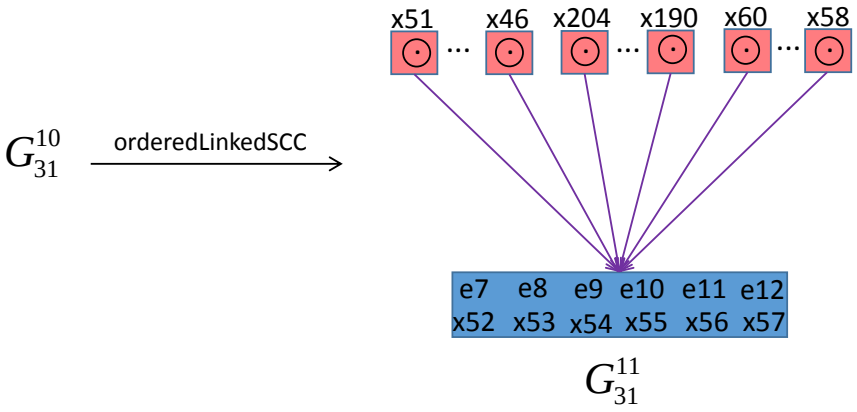


Figure 4.11: Evolution of G_{31}^{1}

Analyzing component 4 (CC4) Component 4 is very similar to component 3. The analysis result shows that equation $e28$ is redundant (type I). It is redundant with $\{e25, e26, e27, e29, e30\}$ which corresponds to the spanning group.

	x46	x47	x48	x49	x50	x51	x52	x53	x54	x55	x56	x57	x58	x59	x60	x190	x191	x192	x193	x194	x195	x196	x197	x198	x199	x200	x201	x202	x203	x204
e7	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
e8	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0
e9	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1
e10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
e11	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
e12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 4.7: Component 3 (CC3 6×30)

	x67	x68	x69	x70	x71	x72	x211	x212	x213	x214	x215	x216
e13	0	0	0	1	0	0	0	0	0	1	0	0
e14	0	0	0	0	1	0	0	0	0	0	1	0
e15	0	0	0	0	0	1	0	0	0	0	0	1
e16	1	1	1	1	1	1	1	1	1	1	1	1
e17	1	1	1	1	1	1	1	1	1	1	1	1
e18	1	1	1	1	1	1	1	1	1	1	1	1
e49	0	0	0	0	0	1	0	0	0	0	0	0
e50	0	0	0	0	0	1	0	0	0	0	0	0

Table 4.8: Component 5 (CC5 8×12)

Analyzing component 5 (CC5) Component 5 contains 8 equations and 12 variables (table 4.8). After D-M decomposition, it is initially decomposed into structurally over-constrained (G_{51}^{10}), well-constrained and under-constrained subpart (table 4.9). For G_{51}^{10}, at the step of G_{51}^{11}, the red block is solvable and the solution is propagated to the block containing equation $e50$. As a result, the difference is -1.40e+01 and it is conflicting with $e49$ (type II). After launching D-M decomposition second time, the system is decomposed into structurally well-constrained (G_{52}^{20}) and under-constrained subparts (table 4.10). The evolution of G_{52}^{20} is shown in figure 4.13. However, there is no over-constraints identified and the solution of the red block are propagated to the other blocks. Finally, after applying D-M decomposition three times, the whole system is structurally under-constrained (table 4.11). Since all red blocks contain only variables, all the equations in table 4.11 are analyzed as a whole. As a result, equation $e16$, $e17$ and $e18$ are redundant (type I). The spanning group of the three equations are $\{e13, e14, e15, e49\}$.

	x211	x212	x213	x214	x215	x67	x68	x69	x70	x71	x216	x72
e16	1	1	1	1	1	1	1	1	1	1	1	1
e17	1	1	1	1	1	1	1	1	1	1	1	1
e18	1	1	1	1	1	1	1	1	1	1	1	1
e13	0	0	0	1	0	0	0	0	1	0	0	0
e14	0	0	0	0	1	0	0	0	0	1	0	0
e15	0	0	0	0	0	0	0	0	0	0	1	1
e49	0	0	0	0	0	0	0	0	0	0	0	1
e50	0	0	0	0	0	0	0	0	0	0	0	1

Table 4.9: First time D-M decomposition on CC5 (d=1)

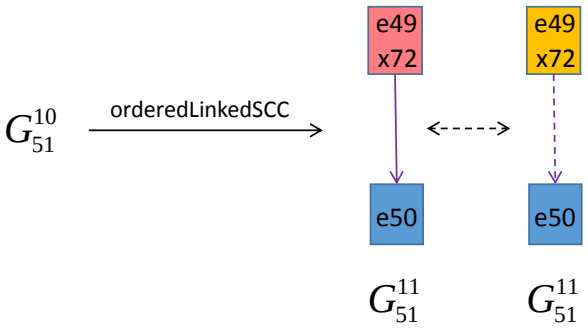


Figure 4.12: Evolution of G_{51}^{1}

	x67	x68	x69	x70	x71	x211	x212	x213	x214	x215	x216
e16	1	1	1	1	1	1	1	1	1	1	1
e17	1	1	1	1	1	1	1	1	1	1	1
e18	1	1	1	1	1	1	1	1	1	1	1
e13	0	0	0	1	0	0	0	0	1	0	0
e14	0	0	0	0	1	0	0	0	0	1	0
e15	0	0	0	0	0	0	0	0	0	0	1

Table 4.10: Second time D-M decomposition on CC5 (d=2)

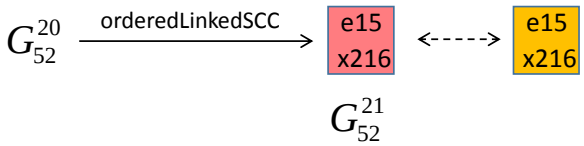


Figure 4.13: Evolution of G_{52}^{2}

	x211	x212	x213	x214	x215	x67	x68	x69	x70	x71
e16	1	1	1	1	1	1	1	1	1	1
e17	1	1	1	1	1	1	1	1	1	1
e18	1	1	1	1	1	1	1	1	1	1
e13	0	0	0	1	0	0	0	0	1	0
e14	0	0	0	0	1	0	0	0	0	1

Table 4.11: Third time D-M decomposition on CC5 (d=3)

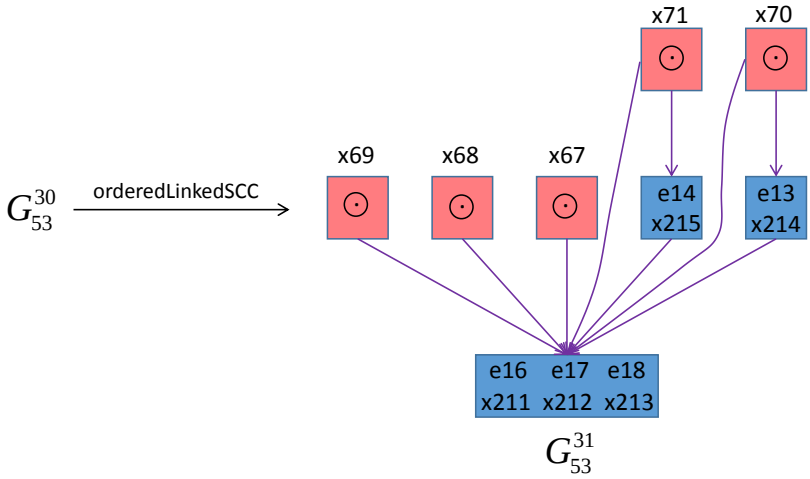


Figure 4.14: Evolution of G_{53}^{1}

Analyzing component 6 (CC6) Component 6 contains 6 equations and 12 variables (table 4.12), which is structurally under-constrained (table 4.13) after applying D-M decomposition. The evolution of G_{63}^{1} is shown in figure 4.15. At the step of G_{63}^{11}, the red blocks contain only variables and thus the algorithm takes all the equations as input and equations $e34$, $e35$ and $e36$ are all redundant with $\{e31, e32, e33\}$ (type I).

	x163	x164	x165	x166	x167	x168	x115	x116	x117	x118	x119	x120
e34	1	1	1	1	1	1	1	1	1	1	1	1
e35	1	1	1	1	1	1	1	1	1	1	1	1
e36	1	1	1	1	1	1	1	1	1	1	1	1
e31	0	0	0	1	0	0	0	0	0	1	0	0
e32	0	0	0	0	1	0	0	0	0	0	1	0
e33	0	0	0	0	0	1	0	0	0	0	0	1

Table 4.12: Component 6 (CC6 6×12)

	x163	x164	x165	x166	x167	x168	x115	x116	x117	x118	x119	x120
e34	1	1	1	1	1	1	1	1	1	1	1	1
e35	1	1	1	1	1	1	1	1	1	1	1	1
e36	1	1	1	1	1	1	1	1	1	1	1	1
e31	0	0	0	1	0	0	0	0	0	1	0	0
e32	0	0	0	0	1	0	0	0	0	0	1	0
e33	0	0	0	0	0	1	0	0	0	0	0	1

Table 4.13: First time D-M decomposition on CC6 (d=1)

Analyzing component 7 (CC7) Component 7 contains 8 equations and 6 variables (table 4.14). After D-M decomposition, it is initially decomposed

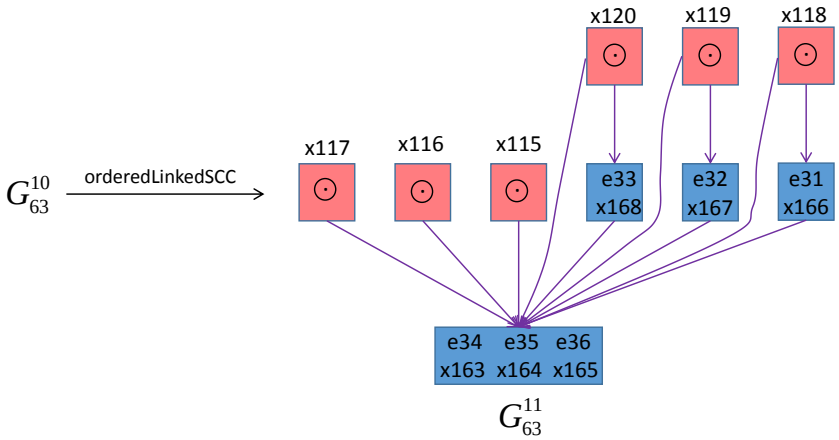


Figure 4.15: Evolution of G_{63}^{1}

into structurally over-constrained (G_{71}^{10}) and well-constrained subpart (table 4.15). For G_{71}^{10}, at the step of G_{71}^{11}, the red block is solvable and the solution is propagated to the block $e52$ and block $e51$. The differences are -2.52 and 9.16 respectively and the two are all conflicting with $e43$ (type II). After launching D-M decomposition second time, the system is structurally well-constrained (G_{72}^{20}). The evolution of G_{72}^{20} is shown in figure 4.17. However, there is no over-constraints identified and solutions of red blocks are propagated to the other blocks.

	x8	x7	x9	x11	x12	x10
e47	1	1	1	1	1	1
e46	0	1	1	0	1	1
e48	0	1	1	0	1	1
e44	0	0	0	1	0	0
e45	0	0	0	0	1	0
e43	0	0	0	0	0	1
e51	0	0	0	0	0	1
e52	0	0	0	0	0	1

Table 4.14: Component 7 (CC7 8×6)

	x8	x7	x9	x11	x12	x10
e47	1	1	1	1	1	1
e46	0	1	1	0	1	1
e48	0	1	1	0	1	1
e44	0	0	0	1	0	0
e45	0	0	0	0	1	0
e43	0	0	0	0	0	1
e51	0	0	0	0	0	1
e52	0	0	0	0	0	1

Table 4.15: First time D-M decomposition on CC7 (d=1)

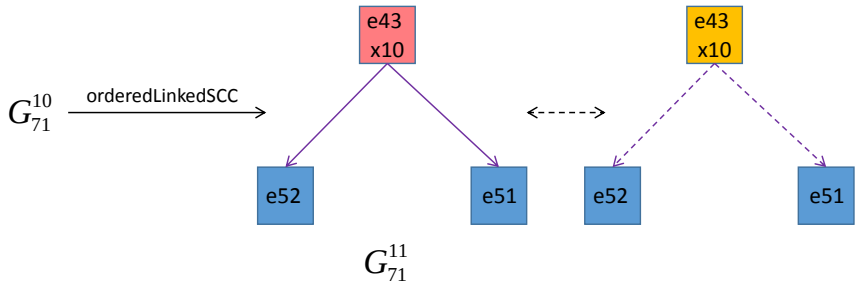


Figure 4.16: Evolution of G_{71}^{1}

	x8	x7	x9	x11	x12
e47	1	1	1	1	1
e46	0	1	1	0	1
e48	0	1	1	0	1
e44	0	0	0	1	0
e45	0	0	0	0	1

Table 4.16: Second time D-M decomposition on CC7 (d=2)

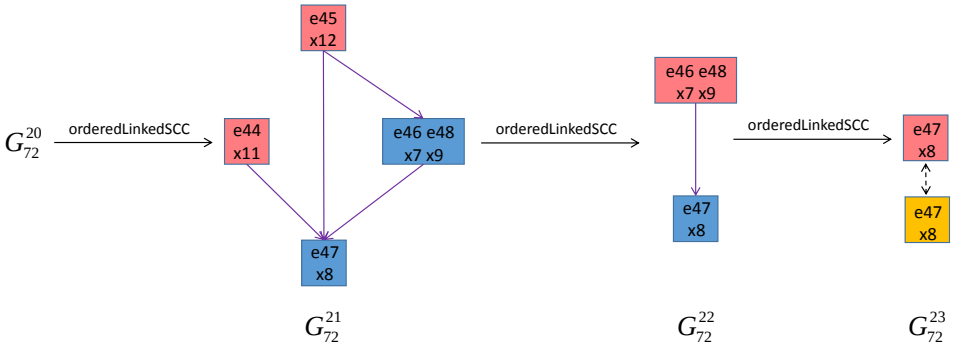


Figure 4.17: Evolution of G_{72}^{1}

Analyzing component 8 (CC8) Component 8 contains 6 equations and 3 variables (table 4.17). After D-M decomposition, it is initially decomposed into structurally over-constrained (G_{81}^{10}) subpart (table 4.18). For G_{81}^{10}, at the step of G_{81}^{11}, the red blocks are solvable and solutions are propagated to the block $e41$, $e42$, and $e40$. The differences are -2.94e-7, –5.49e-12, and -1.29e-11 respectively and they are redundant with $\{e37, e39\}$, $\{e37, e38, e39\}$, and $\{e37, e39\}$ respectively (type II).

	x1	x2	x3
e37	1	0	0
e38	0	1	0
e39	0	0	1
e40	1	0	1
e41	1	1	1
e42	1	0	1

Table 4.17: Component 8 (CC8 6×3)

	x1	x2	x3
e37	1	0	0
e38	0	1	0
e39	0	0	1
e40	1	0	1
e41	1	1	1
e42	1	0	1

Table 4.18: First time D-M decomposition on CC8 (d=1)

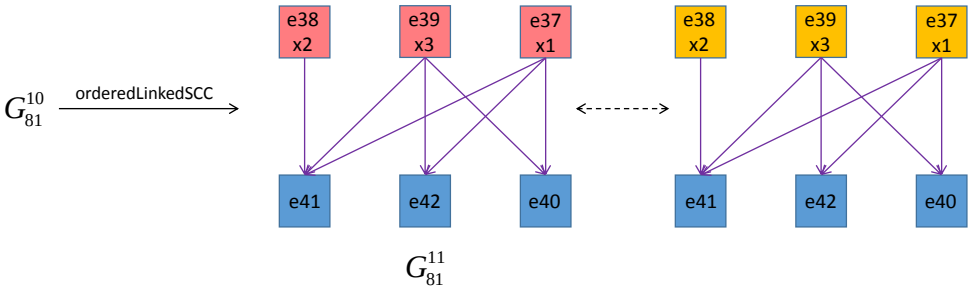


Figure 4.18: Evolution of G_{81}^{1}

Finally, the proposed detection and resolution method have been applied on 8 component and the above detection results are summarized in the table 4.19.

Component	Initial D-M			D-M	Over-constraints			
	Over	Well	Under	times	Redundant	Type	Conflicting	Type
1			√	1	$e6$	I		
2			√	1	$e24$	I		
3			√	1	$e10$	I		
4			√	1	$e28$	I		
5	√	√	√	3	$e16/e17/e18$	I	$e50$	II
6			√	1	$e34/e35/e36$	I		
7	√	√		2			$e51/e52$	II
8	√			1	$e40/e41/e42$	II		

Table 4.19: Detection results of the teapot geometry with non-linear equations directly

4.2.3 Over-constraints and the spanning groups

Detected over-constraints and the corresponding spanning groups are summarized in table 4.20. It shows which equation is redundant/conflicting with which group of equations. However, it cannot be presented to users for debugging purpose directly since users are not directly working at the level of the equations but at the level of the constraints. To make it easier to understand, over-constraints and the spanning groups at the level of geometry is shown in table 4.21. The correspondences between constraints and equations is given in table 4.5.

Over-constraint	Spanning Group
e6	e1, e2, e3, e4, e5
e24	e19, e20, e21, e22, e23
e10	e7, e8, e9, e11, e12
e28	e25, e26, e27, e29, e30
e50	e49
e16	e13, e14, e15, e49
e17	e13, e14, e15, e49
e18	e13, e14, e15, e49
e34	e31, e32, e33
e35	e31, e32, e33
e36	e31, e32, e33
e51	e43
e52	e43
e40	e37, e39
e41	e37, e38, e39
e41	e37, e38, e39
e42	e37, e39

Table 4.20: Over-constraint and the spanning group at the level of equations on the teapot

Over-constraint	Spanning Group
constraint 1	constraint 2
constraint 8	constraint 7
constraint 4	constraint 3
constraint 10	constraint 9
constraint 18	constraint 17
constraint 6	constraint 5, 17
constraint 12	constraint 11
constraint 19	constraint 15
constraint 20	constraint 15
constraint 14	constraint 13

Table 4.21: Over-constraints and the spanning groups at the level of geometries on the teapot

4.2.4 Result of linearizion

Linearizion In the work of Moinet (Moinet, Mandil, and Serre, 2014), and Serrano (Serrano, 1987), non-linear constraints systems are linearized so that linear detection methods can be applied. The linearizion is based on Taylor-series expansion at a given point and the linear detection methods are QR (used by Hu et al) and G-J (used by Moinet, Serrano).

To know whether linearizion of non-linear system affects the detection results or not, equations system of teapot geometry are linearized at the witness configuration starting from initial sketch. Then, our algorithm is used to detect the numerical over-constraints. The results are summarized in table 4.22. Comparing to the table 4.19, $\{e34, e35\}$ are detected as conflicting and $\{e46, e48\}$ are new conflicting constraints.

Component	Initial D-M			D-M	Over-constraints			
	Over	Well	Under	times	Redundant	Type	Conflicting	Type
1			√	1			$e6$	I
2			√	1			$e24$	I
3			√	1			$e10$	I
4			√	1			$e28$	I
5	√	√	√	3	$e16/e17/e18$	I	$e50$	II
6			√	1	$e36$	I	$e34/e35$	I
7	√	√		2			$e51/e52, e46/e48$	II,I
8	√			1	$e40/e41/e42$	II		

Table 4.22: Detection results of linearized teapot geometry

In our algorithm, a redundant and conflicting constraint is distinguished by comparing the difference $\Delta|P_0 - P'|$ of the over-constraint between initial specified value ($e(X_0) = P_0$) and final value ($e(X') = P'$) after releasing over-constraints with defined tolerance. The value of the tolerance used in this testing case is $1e$-4. The deviations (Δ) from the design intent after removing the identified over-constraints are summarized in table 4.23 for both the original system and the linearized one.

From the table, it is obvious that differences of $\{e6, \cdots, e36\}$ of linearized system are much higher than those of original system. This is because the linearized system ignores the truncation error $O(X^2)$ when expanding the Taylor series at the witness to the first order. Meanwhile, it can cause independent constraints detected as over-constraints, which may be wrong in some cases ($\{e46, e48\}$, for example). Therefore, linearizion of non-linear system is unreliable both in detecting numerical over-constraints as well as distinguishing redundant and conflicting constraints. As a result, linearizion of non-linear systems is not recommended when dealing with non-linear systems.

Original		Linearized	
over-constraint	Δ	over-constraint	Δ
e6	1.32e-14	e6	0.0118
e24	6.46e-15	e24	0.0111
e10	5.31e-15	e10	4.55e-04
e28	2.91e-14	e28	0.0017
e16	3.10e-09	e16	3.69e-05
e17	1.70e-09	e17	2.01e-05
e18	1.11e-13	e18	1.47e-08
e34	2.68e-14	e34	0.0096
e35	2.17e-14	e35	0.0052
e36	1.35e-14	e36	3.97e-07
e40	-1.29e-11	e40	-1.29e-11
e41	2.94e-07	e41	2.94e-07
e42	-5.46e-12	e42	-5.46e-12
e50	-1.40e+01	e50	-1.40e+01
e51	-2.52	e51	-2.52
e52	9.16	e52	9.16
		e46	0.0074
		e48	0.0034

Table 4.23: Deviations (Δ) from the design intent when removing over-constraints detected in original and linearized systems.

This testing example has shown that our algorithm works well on free form configurations. More specifically, it can deal properly with the local support property. However, the detection is still at the level of equations and the resolution of geometric over-constraints is not addressed. All these issues will be discussed in a 3D glass geometry in the next section.

4.3 Sketching a 3D glass

In this example, the idea is to show how the proposed over-constraints detection and resolution approach can support the sketching of a 3D glass composed of 4 connected NURBS patches. The designer sketches his/her design intent and associated requirements.

4.3.1 Modeling

Here, the objective is to modify the upper part of the glass by specifying the following elements:

- *Variables*: Each patch has a degree 5×5 and has a control polygon made of 16×6 control points which coordinates are the variables of our optimization process (figure 4.19a). Since the objective is to modify the upper part of the glass, the designer selects how many rows of control

points are to be blocked and how many can move. For example, if the designer wishes to free 4 upper rows of control points of the four patches, then there will be $4 \times (6 \times 4) \times 3 = 288$ variables in the unknown vector X. The results will be illustrated with 4 and 5 rows free to move.

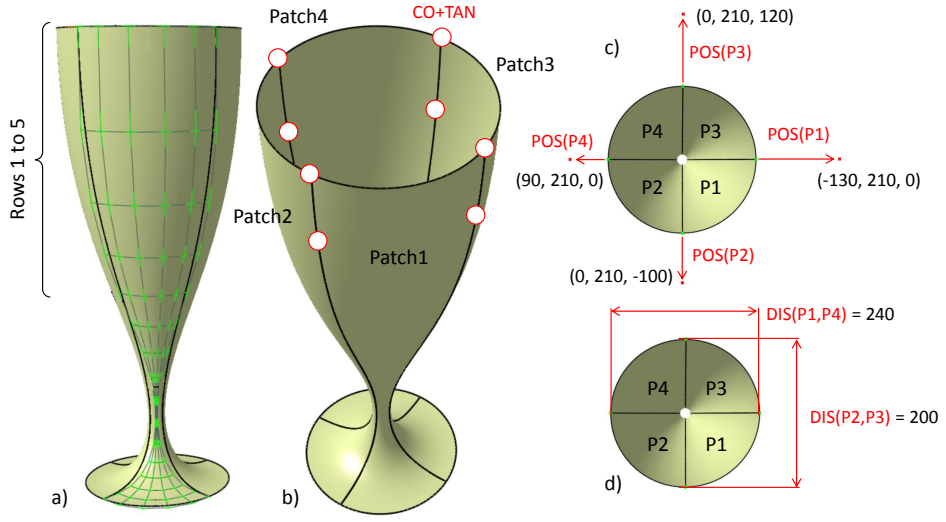


Figure 4.19: Initial sketch of glass geometry.

- *Constraints*: Three types of constraints are used to specify how the shape of the 3D glass has to evolve:

  - *Position*: 4 position constraints are added to the four end points of the patches along the upper boundary curves. As shown in figure 4.19c, the green points of the patches need to move to new positions in 3D space. They are labeled from 1 to 4 and they generate $4 \times 3 = 12$ linear equations labeled from 1 to 12 (table 4.24).

  - *Distance*: 2 distance constraints are defined between the opposite sides of the patches (figure 4.19d). They are labeled 5 and 6 and they generate 2×1 non-linear equations labeled 13 and 14 (table 4.24).

  - *Coincidence and tangency*: 8 coincidence and 8 tangency constraints are specified to maintain the continuity between the upper parts of the patches during the deformation (figure 4.19b). They are labeled from 7 to 22 and they generate 8×3 linear and 8×3 non-linear equations labeled from 15 to 62 (table 4.24). The non-linearity comes from the use of the vector product to express the collinearity of normals.

Overall, there are 22 geometric constraints generating 62 equations in the set $F(X) = 0$. Some of those constraints are conflicting and it is the purpose of this section to try to see how our algorithm can detect and remove them without affecting too much the design intent.

- *Objective function*: Since the proposed approach removes the identified over-constraints, the resulting system of equations $\widetilde{F}_b(X) = 0$ (section 3.3) may become under-constrained and a function $G(X)$ has to be minimized. Here, the idea is to make use of the approach of (Pernot et al., 2005) to define two types of deformation behavior: either a minimization of the variation of the shape ($G_1(X)$) between the initial and final configurations, or minimization of the area of the final shape ($G_2(X)$). In terms of design intent, the first one tends to preserve the initial shape of the glass, whereas the second forgets the initial shape and tends to generate surfaces similar to tensile structures.

Constraint	Equations	Type	Component
4	1-3	linear	1
2	4-6	linear	2
1	7-9	linear	1
3	10-12	linear	2
5	13	non-linear	1
6	14	non-linear	2
7	15-17	linear	1
8	18-20	non-linear	1
9	21-23	linear	1
10	24-26	non-linear	1
11	27-29	linear	2
12	30-32	non-linear	2
13	33-35	linear	2
14	36-38	non-linear	2
15	39-41	linear	1
16	42-44	non-linear	1
17	45-47	linear	1
18	48-50	non-linear	1
19	51-53	linear	2
20	54-56	non-linear	2
21	57-59	linear	2
22	60-62	non-linear	2

Table 4.24: Typology of constraints and equations involved in the description of the 3D glass sketching example

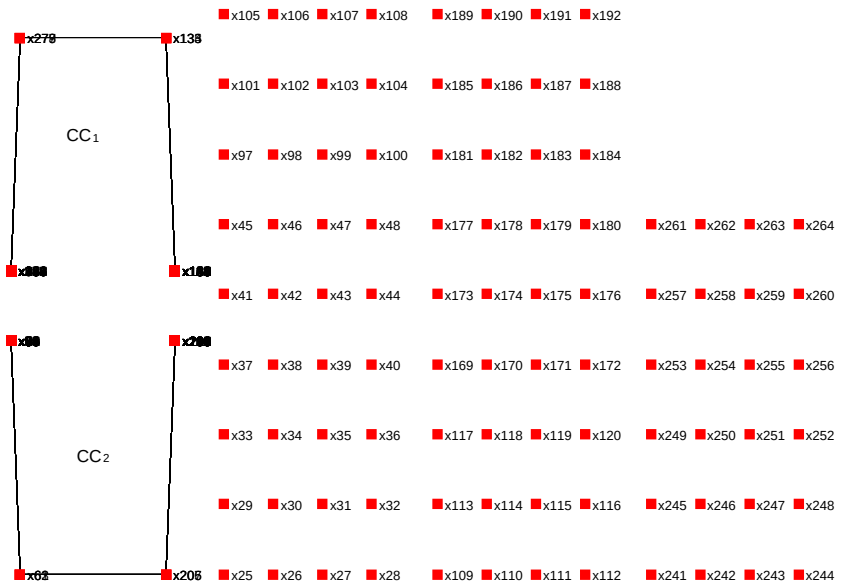


Figure 4.20: Constraint graph between variables

4.3.2 Detection Process

Finding local parts

After applying BFS, as visible in figure 4.20, there are 98 unconnected components (red vertices) in the graph with 96 of them containing only variables (right part of the figure). The other two components CC_1 and CC_2 contain both variables and equations. Since the two components are similar, the process is illustrated only with CC_2 component.

Analysis of the local parts

After applying D-M decomposition (1st time) on initial CC_2^0, we got structurally over-constrained subpart (G_{21}^1, green section of table 4.26), well-constrained subpart (G_{22}^1, yellow section of table 4.26) and under-constrained subpart (G_{23}^1, gray section of table 4.26). Maximum matching of these subparts is the diagonal (marked red) of each section.

The process of analyzing each subpart is illustrated as following:

- Evolution of G_{21}^1 subpart. As it is shown in figure 4.21, initially G_{21}^1 is set to G_{21}^{10} before maximum matching and strong connected blocks are generated. After applying *orderedLinkedSCC*, strong connected components $\{SCC_{211}^{11}, \cdots, SCC_{215}^{11}\}$ are generated as red blocks and G_{21}^{10} is now updated to G_{21}^{11}. Since these blocks are solvable, the solutions are

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x133	x134	x135	x136	x137	x138	x139	x140	x141	x142	x143	x144	x67	x68	x69	x61	x62	x63	x64	x65	x66	x70	x71	x72	x73	x74	x75	x76	x77	x78	x79	x80	x81	x82	x83	x84
e4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
e28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
e29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
e30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
e31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
e32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
e33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
e34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	0	0
e35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	0	0
e36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
e37	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
e38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
e51	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e52	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e53	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e54	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e55	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e56	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e57	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e58	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e59	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e60	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e61	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e62	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4.25: Component 2 (CC2 31 × 48)

	x12	x133	x134	x135	x136	x137	x138	x142	x143	x65	x66	x78	x79	x80	x81	x82	x83	x84	x4	x5	x6	x7	x8	x9	x10	x11	x144	x61	x62	x63	x64	x70	x71	x72	x76	x77	x1	x2	x3	x73	x74	x75	x139	x140	x141	x67	x68	x69
e57	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
e58	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
e55	0	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0
e56	0	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0
e61	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0
e54	0	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0
e60	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0
e62	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	1	1	0	0	0
e59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0
e36	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1
e32	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1
e38	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1
e37	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1
e33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0
e34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0
e35	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1
e31	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1
e30	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	1	1	0	0	0	1	1	1	0	0	0	1	1	1
e51	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
e52	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0
e53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
e27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
e28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0
e29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
e10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
e11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
e12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
e4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
e5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
e6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
e14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1

Table 4.26: First time D-M decomposition on CC2 (d=1)

propagated to the other dependent blocks. Same procedure is applied on G_{21}^{11}, where the red block is solved and the solution is fed to the component containing $e14$. As a result, it is conflicting and the spanning group is $\{e4, e5, e6, e10, e11, e12\}$. $e14$ is removed and the system is now updated to CC_2^1.

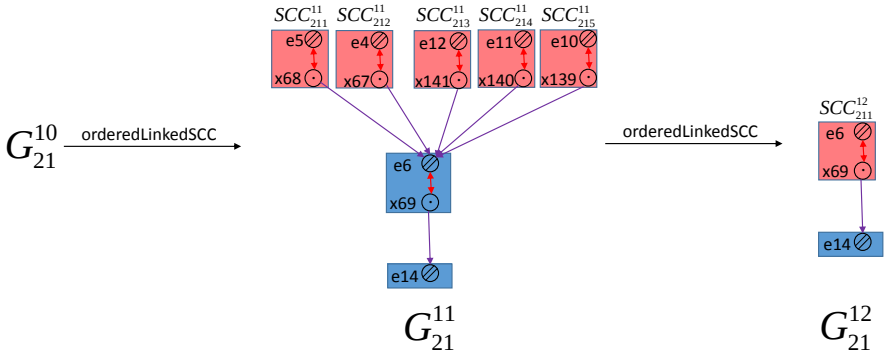


Figure 4.21: Evolution of G_{21}^1

- Evolution of G_{22}^2 subpart. For the new system CC_2^1, D-M is applied second time resulting a structurally well-constrained subpart (G_{22}^2, yellow section of the table 4.27) and under-constrained subpart (G_{23}^2, gray section of the table 4.27). As it is shown in figure 4.22, after applying *orderedLinkedSCC* to G_{22}^{20}, strong connected components $\{SCC_{221}^{21}, \cdots, SCC_{226}^{21}\}$ are solvable red blocks. No numerical over-constraints are found and solutions are propagated to the other dependent blocks. The system is now updated to CC_2^2.

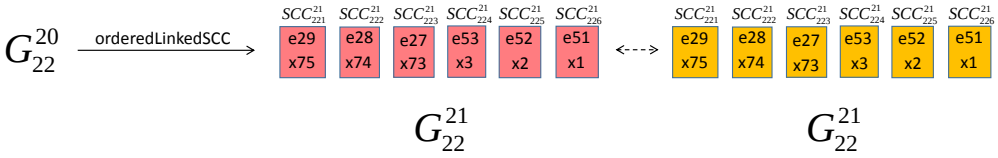


Figure 4.22: Evolution of G_{22}^2

- Evolution of G_{23}^3 subpart. After employing D-M decomposition third time to the new system CC_2^2, only structurally under-constrained subpart G_{23}^3 is generated. As it is shown in figure 4.23, after applying *orderedLinkedSCC* on G_{23}^{30}, all the generated red blocks contain only variables. Equations of these blocks are analyzed together. As a result, equations $\{e32, e56, e38, e62\}$ are found to be redundant. The spanning group of each equation is $\{e10, e11, e12, e27, e28, e29, e30, e31, e33, e34, e35, e36, e37, e4, e5, e6, e51, e52, e53, e54, e55, e57, e58, e59, e60, e61\}$.

	x138	x82	x83	x84	x4	x5	x7	x8	x9	x10	x11	x144	x61	x62	x63	x64	x76	x77	x12	x133	x134	x135	x136	x137	x142	x143	x65	x66	x78	x79	x80	x81	x6	x70	x71	x72	x1	x2	x3	x73	x74	x75
e61	1	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0
e55	0	0	0	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	1	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
e56	0	0	0	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	1	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
e54	0	0	0	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	1	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
e60	1	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
e62	1	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	1	0	0	0	1	1	1	1	0	0
e57	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
e58	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
e36	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1	1	1	0	0	0	1	1	1
e38	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1	1	1	0	0	0	1	1	1
e32	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1	0	0	0	1	1	1
e37	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	1	1	1	0	0	0	1	1	1
e31	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1	0	0	0	1	1	1
e30	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1	0	0	0	1	1	1
e59	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
e33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0
e34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
e35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	1
e51	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
e52	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
e53	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
e27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
e28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
e29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 4.27: Second time D-M decomposition on CC2 (d=2)

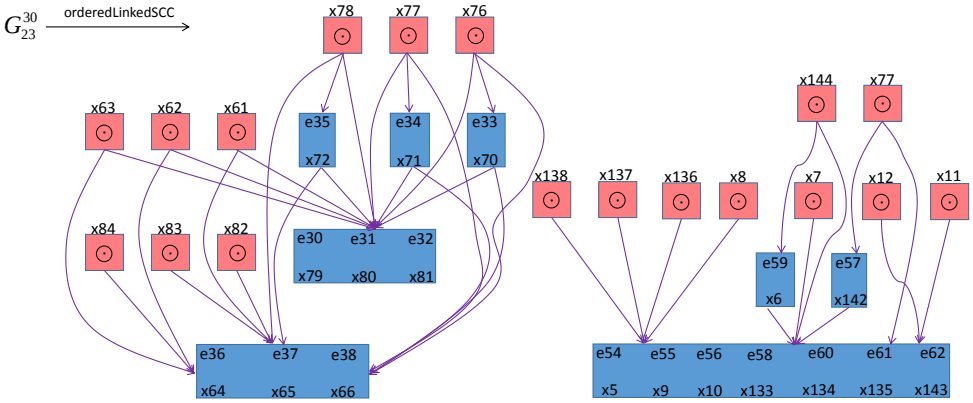


Figure 4.23: Evolution of G_{23}^{3}

The detection results of CC2 are summarized in the table 4.29.

Component	Initial D-M			D-M	Over-constraints			
	Over	Well	Under	times	Redundant	Type	Conflicting	Type
2	√	√	√	3	$e32/e56/e38/e62$	I	$e14$	II

Table 4.29: Detection results of CC2 of the glass geometry

	x10	x11	x64	x12	x133	x134	x135	x136	x137	x65	x66	x79	x80	x81	x6	x70	x71	x72	x138	x82	x83	x84	x4	x5	x7	x8	x9	x144	x61	x62	x63	x76	x77	x142	x143	x78
e61	1	1	0	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	1	0
e36	0	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1	0	1	1	1	0	0	0	0	0	0	1	1	1	1	1	0	0	1
e38	0	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1	0	1	1	1	0	0	0	0	0	0	1	1	1	1	1	0	0	1
e37	0	0	1	0	0	0	0	0	0	1	1	1	1	1	0	1	1	1	0	1	1	1	0	0	0	0	0	0	1	1	1	1	1	0	0	1
e55	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	1	0
e56	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	1	0
e54	0	0	0	0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	1	0
e60	1	1	0	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	1	0
e62	1	1	0	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	1	0
e59	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
e32	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1
e31	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1
e30	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	1
e33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
e34	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
e57	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0
e58	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0
e35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 4.28: Third time D-M decomposition on CC2 (d=3)

4.3.3 Over-constraints and the spanning groups

The analysis of those two components gives rise to the identification of 2 conflicting equations ($e13, e14$) which correspond to either the position or distance constraints. Also, 8 redundant equations are detected, which are contained in 8 tangent constraints (section 4.3.1).

These over-constraints and the corresponding spanning groups at the level of equations is summarized in table 4.30, at the level of constraints is summarized in table 4.31.

Component	Over-constraint	Spanning group
CC2	constraint 6	constraint 2, 3
	constraint 12	constraint 2, 3, 11, 13, 19, 21
	constraint 14	constraint 2, 3, 11, 13, 19, 21
	constraint 20	constraint 2, 3, 11, 13, 19, 21
	constraint 22	constraint 2, 3, 11, 13, 19, 21
CC1	constraint 5	constraint 1, 4
	constraint 8	constraint 1, 4, 7, 9, 15, 17
	constraint 10	constraint 1, 4, 7, 9, 15, 17
	constraint 16	constraint 1, 4, 7, 9, 15, 17
	constraint 18	constraint 1, 4, 7, 9, 15, 17

Table 4.31: Over-constraint and the spanning group at the level of geometries on the glass

Since the result of the detection process is not unique, 9 configurations are obtained and are gathered together in Table 4.32. Here, one has to remember that even if the detection process identifies conflicting equations, our algorithm removes the constraints associated to those equations. For example, configuration 1 considers that the two distance constraints (one between patches P1 and P4, and the other between P2 and P3) are to be removed (0 in the table) and the 4 position constraints are kept (1 in the table).

Config.	DIS(P1,P4)	DIS(P2,P3)	POS(P1)	POS(P2)	POS(P3)	POS(P4)
1	0	0	1	1	1	1
2	1	0	0	1	1	1
3	1	0	1	1	1	0
4	0	1	1	0	1	1
5	0	1	1	1	0	1
6	1	1	0	0	1	1
7	1	1	1	1	0	0
8	1	1	1	0	1	0
9	1	1	0	1	0	1

Table 4.32: Status of the distance and position constraints (0 to remove and 1 to keep) to solve the 9 over-constrained configurations

Component	Over-constraint	Spanning group
CC2	e14	e4, e5, e6, e10, e11, e12
	e32	e10, e11, e12, e27, e28, e29, e30, e31, e33, e34, e35, e36, e37, e4, e5, e6, e51, e52, e53, e54, e55, e57, e58, e59, e60, e61
	e38	e10, e11, e12, e27, e28, e29, e30, e31, e33, e34, e35, e36, e37, e4, e5, e6, e51, e52, e53, e54, e55, e57, e58, e59, e60, e61
	e56	e10, e11, e12, e27, e28, e29, e30, e31, e33, e34, e35, e36, e37, e4, e5, e6, e51, e52, e53, e54, e55, e57, e58, e59, e60, e61
	e62	e10, e11, e12, e27, e28, e29, e30, e31, e33, e34, e35, e36, e37, e4, e5, e6, e51, e52, e53, e54, e55, e57, e58, e59, e60, e61
CC1	e13	e1, e2, e3, e7, e8, e9
	e18	e1, e2,e3, e7, e8, e9, e15, e16, e17, e19, e20, e21, e22, e23, e25, e26, e39, e40, e41, e43, e44, e45, e46, e47, e49, e50
	e24	e1, e2,e3, e7, e8, e9, e15, e16, e17, e19, e20, e21, e22, e23, e25, e26, e39, e40, e41, e43, e44, e45, e46, e47, e49, e50
	e42	e1, e2,e3, e7, e8, e9, e15, e16, e17, e19, e20, e21, e22, e23, e25, e26, e39, e40, e41, e43, e44, e45, e46, e47, e49, e50
	e48	e1, e2,e3, e7, e8, e9, e15, e16, e17, e19, e20, e21, e22, e23, e25, e26, e39, e40, e41, e43, e44, e45, e46, e47, e49, e50

Table 4.30: Over-constraint and the spanning group at the level of equations on the glass

All configurations are then solved while acting on both the number of upper rows to be fixed ($N_{rows} = 4$ or 5), and the objective function to be minimized (either $G_1(X)$ or $G_2(X)$. The effects of applying two objective functions on configuration 1 are shown in figure 4.25 a). The results are gathered together in Tables 4.33 and 4.34. Each configuration is evaluated through the three previously introduced criteria *dg*, *df* and *cond*. Some solutions are shown in Figure 4.24.

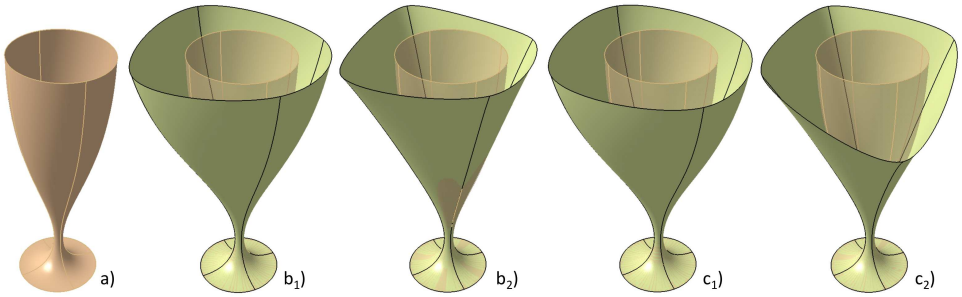


Figure 4.24: Results of the sketching after removing conflicting constraints with $N_{rows} = 4$: (a) initial glass, (b_1) configuration 1 and minimization of the shape variation, (b_2) configuration 1 and minimization of the area of the final surface, (c_1) configuration 3 and minimization of the shape variation, (b_2) configuration 3 and minimization of the area of the final surface

One can first notice that depending on the configurations, the deviation *df* on the constraints varies. For example, with $N_{rows} = 4$ and while minimizing $G_1(X)$, the configuration 7 generates a solution that is closer to the design intent than configuration 6 ($0.10684 < 0.12607$ in Table 4.33). For configuration 3, it is clear that the deviation to the design intent in terms of constraints is more important when minimizing the area of the final surface than when minimizing the shape variation ($0.2288 > 0.10179$ in Table 4.33). This is clearly visible on Figures 4.24c_1 and 4.24c_2.

But the deviation dg_i on the objective function to be minimized also varies. While considering the minimization of the shape variation, one can see that configuration 3 is less interesting than configuration 1 in the sense that it minimizes less the shape variation ($15459.52 > 13801.04$ in Table 4.33).

Finally, for a given configuration, one can notice that when the number of free rows increases, i.e. when there is more freedom, the objective function decreases and the solution is therefore closer to the design intent. This is visible when comparing values from Tables 4.33 and 4.34. For example, in figure 4.25 b), the shape variation of configuration 1 of 5 rows is more minimized than the one of 4 rows ($11266.93 < 13801.04$). Thus, the selection

of the variables X are also important when setting up the optimization problem, which should affect the design intent.

	Minimization of $G_1(X)$			**Minimization of $G_2(X)$**		
Config.	dg_1	df	$cond$	dg_2	df	$cond$
1	13801.04	0.10000	2.8654e19	96733.72	0.10000	1.4272e18
2	17990.88	0.10182	8.3172e18	95225.05	0.28157	4.3894e17
3	15459.52	0.10179	1.5071e19	94483.08	0.22880	4.9533e17
4	12265.51	0.10975	8.8857e18	89924.13	0.22806	3.9399e18
5	10970.98	0.10971	3.9852e19	86879.47	0.25225	8.2501e18
6	15826.68	0.12607	3.4260e18	76878.26	0.68278	1.6567e18
7	12936.45	0.10465	3.8205e18	76167.99	0.62820	3.9842e17
8	13889.18	0.11385	2.5681e18	78657.81	0.59160	4.1485e18
9	14883.21	0.11720	1.2523e18	74351.81	0.71765	6.7658e16

Table 4.33: Evaluation of the 9 configurations with $N_{rows} = 4$

	Minimization of $G_1(X)$			**Minimization of $G_2(X)$**		
Config.	dg_1	df	$cond$	dg_2	df	$cond$
1	11266.93	0.10000	4.0149e17	85121.36	0.10000	3.3441e17
2	14719.05	0.10280	4.6031e17	86295.47	0.25034	6.5355e19
3	12506.55	0.10277	1.7748e19	85190.96	0.20076	1.0972e18
4	9944.87	0.11452	1.7903e18	79428.31	0.20592	1.7041e18
5	8799.29	0.11448	6.1454e17	77800.57	0.22919	1.0218e18
6	12561.66	0.13935	4.1681e18	69603.16	0.76646	8.5100e16
7	10441.11	0.10684	1.0862e18	69502.72	0.70009	2.3460e18
8	11134.09	0.12097	2.5394e18	71465.72	0.65901	1.5773e18
9	11877.59	0.12601	1.3790e19	67661.55	0.80372	8.4472e17

Table 4.34: Evaluation of the 9 configurations with $N_{rows} = 5$

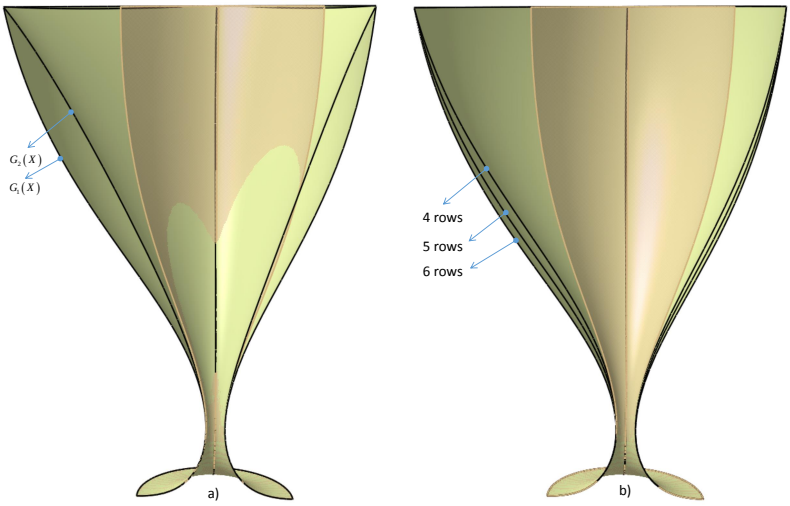


Figure 4.25: a) Configuration 1 with 5 free rows after minimizing G_1 and G_2; b) Configuration 1 with 4,5,6 free rows after minimizing G_1

4.4 Additional experiments

4.4.1 Effect of system decomposition on computation time

We implemented the Algorithm 1 and 2 (Section 3.2.3) in MATLAB. Our code was executed on a Dell Precision M4800, Windows 7 system. To know the effect of system decomposition, experiments are conducted on the glass example and the results are compared with a detection method without decomposition. This method used consists in $WCM + IS$ (section 3.2.2) shown in figure 4.26.

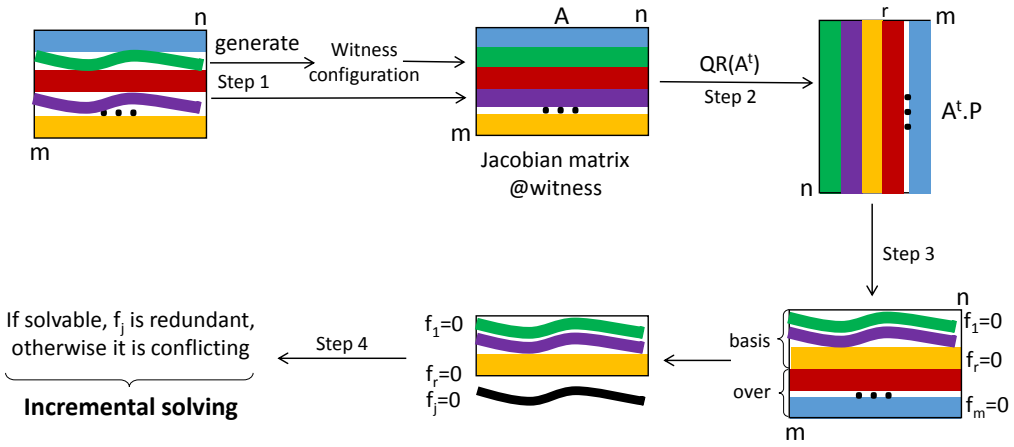
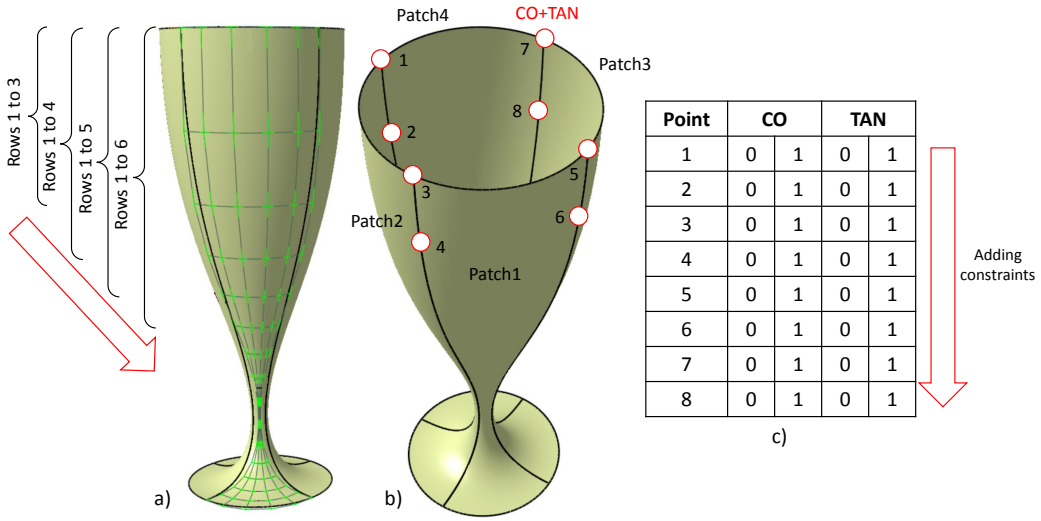


Figure 4.26: Method $WCM + IS$ without decomposition

Experiments are based on a glass series with different number of the variables and constraints (coincidence and tangent constraints). As it is shown in figure 4.27, variables are the coordinates of control points ranging from row 1 to row 3,4,5 and 6 with corresponding variables of number 216,288,360 and 432 respectively. Also, the coincidence and tangent constraints of point 1-8 are added incrementally. In other words, they are turned off (marked 0) at the beginning. Then, they are added incrementally from point 1 to 8 by turning on a constraint (marked 1) until all the constraints are added (all the constraints are marked red).



Point	CO		TAN	
1	0	1	0	1
2	0	1	0	1
3	0	1	0	1
4	0	1	0	1
5	0	1	0	1
6	0	1	0	1
7	0	1	0	1
8	0	1	0	1

Figure 4.27: Changing variables and constraints. a) Increasing number of variables b) Increasing number of constraints (coincidence and tangent)

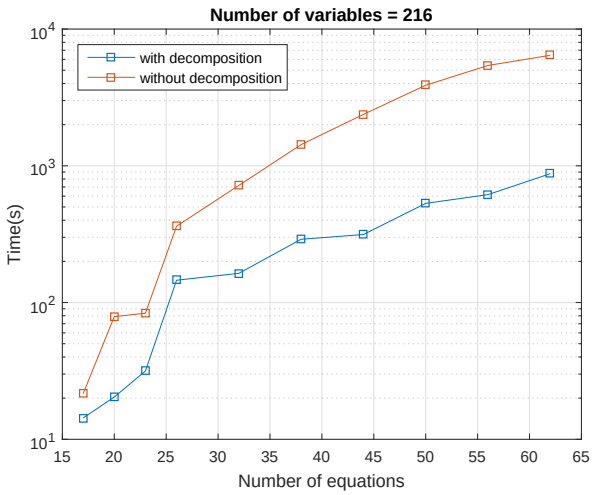


Figure 4.28: Computation time with respect to number of equations when number of variables = 216.

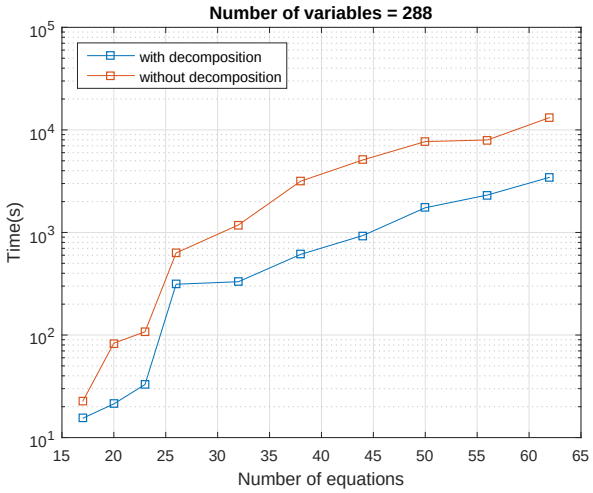


Figure 4.29: Computation time with respect to number of equations when number of variables = 288.

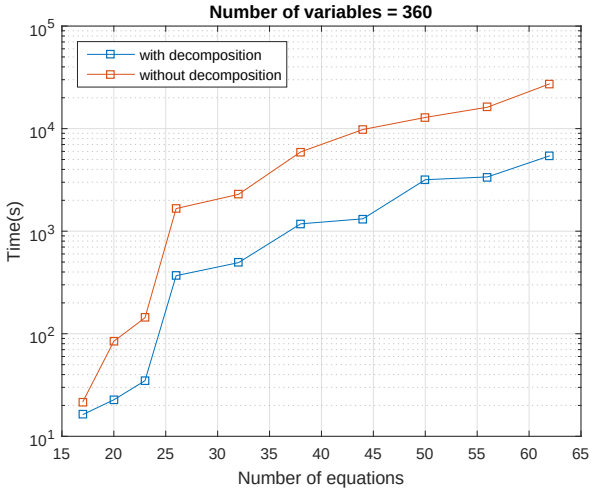


Figure 4.30: Computation time with respect to number of equations when number of variables = 360.

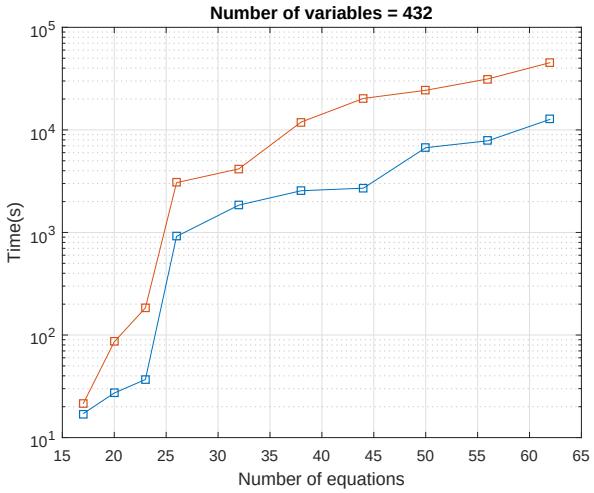


Figure 4.31: Computation time with respect to number of equations when number of variables = 432.

Computation time with respect to 3,4,5 and 6 rows of variables are shown in the figures 4.28, 4.29, 4.30, and 4.31 respectively. For a given set of variables, if the number of equations are less than 20, the effect of system decomposition is not obvious; However, when adding more equations, system decomposition significantly reduces computational time. The average computation time of figures 4.28, 4.29, 4.30, and 4.31 using decomposition are 6 times faster than ones without decomposition. Since it strongly depends on the system to be analyzed, a theoretical complexity analysis has not yet been done. The reader can anyhow check section 3.2.4 for a first understanding of complexity issues without considering solving process.

4.4.2 Results with respect to tolerance

In our algorithm, the separation between basis (E_b) and over-constraints (E_o) is based on the rank computation of the Jacobian matrix at the witness configuration, where the number of E_b is equal to the value of the rank. However, the latter depends on a tolerance. Actually, we use Singular Value Decomposition (SVD) to compute the rank, which is equal to the number of singular values that are larger than the tolerance (tol_{rank}).

Formally, the singular value decomposition of a $m \times n$ real or complex matrix M is a factorization of the form $U\Sigma V$, where U is a $m \times m$ real or complex unitary matrix, Σ is a $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and V is a $n \times n$ real or complex unitary matrix. The diagonal entries σ_i of Σ are known as the singular values

of M. The rank of M equals to the number of $\sigma_i s > tol_{rank}$ in Σ.

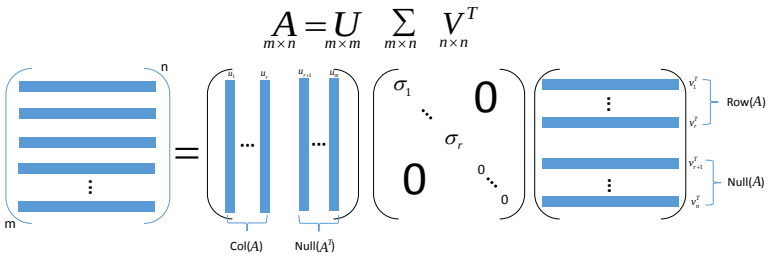


Figure 4.32: Singular Value Decomposition

Moreover, tolerance could also affect the number of conflicting and redundant constraints. In our algorithm, the detected over-constraints are of two types: type I and type II. Further distinguishing on the redundant and conflicting constraints inside the two types of over-constraints requires a solving and feeding process respectively. On one hand, termination of a solving process is determined by termination tolerance on the function value, on the first-order optimality, and on the values of variables between steps. On the other hand, substituting a solution(X_0) to a constraint($e(X) = P$) would give new value to the constraint($e(X_0) = P'$). Comparing the absolute difference between the two ($|P - P'|$) with the tolerance would determine if a constraint is conflicting ($|P - P'| >$ tolerance) or redundant($|P - P'| \leq$ tolerance). Here, we define this tolerance as tol_{rc}.

In this section, we use the teapot example to show the variation of the two type of tolerances (tol_{rc} and tol_{rank}) on the results of the over-constraints detection. The tol_{rc}, tol_{rank} are ranging from 10^{-1} to 10^{-10} respectively. Note that, in the teapot testing case (section 4.2), the two tolerances are both fixed to 10^{-6}. The results are summarized in the tables 4.35, 4.36, and 4.37. The results of the testing case in section 4.2 are highlight in tables below.

$10^{?}$	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
-1	13	13	13	9	7	7	7	6	3	3
-2	13	13	13	8	7	7	7	3	3	3
-3	13	13	13	9	7	7	7	3	3	3
-4	13	13	13	9	7	7	7	3	3	3
-5	13	13	13	8	7	7	7	3	3	3
-6	13	13	13	9	7	7	7	3	3	3
-7	13	13	13	8	7	7	7	3	3	3
-8	13	13	13	8	7	7	7	4	3	3
-9	13	13	13	9	7	7	7	3	3	3
-10	13	13	13	9	7	7	7	4	3	3

Table 4.37: Number of identified redundant constraints with respect to tol_{rank}(column) and tol_{rc}(row) on the teapot

10^i	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
-1	16	16	16	12	10	10	10	6	6	6
-2	16	16	16	11	10	10	10	6	6	6
-3	16	16	16	12	10	10	10	6	6	6
-4	16	16	16	12	10	10	10	6	6	6
-5	16	16	16	11	10	10	10	6	6	6
-6	16	16	16	12	10	10	10	6	6	6
-7	16	16	16	11	10	10	10	6	6	6
-8	16	16	16	11	10	10	10	7	6	6
-9	16	16	16	12	10	10	10	6	6	6
-10	16	16	16	12	10	10	10	7	6	6

Table 4.35: Number of identified numerical over-constraints with respect to tol_{rank}(column) and tol_{rc}(row) on the teapot

10^i	-1	-2	-3	-4	-5	-6	-7	-8	-9	-10
-1	3	3	3	3	3	3	3	3	3	3
-2	3	3	3	3	3	3	3	3	3	3
-3	3	3	3	3	3	3	3	3	3	3
-4	3	3	3	3	3	3	3	3	3	3
-5	3	3	3	3	3	3	3	3	3	3
-6	3	3	3	3	3	3	3	3	3	3
-7	3	3	3	3	3	3	3	3	3	3
-8	3	3	3	3	3	3	3	3	3	3
-9	3	3	3	3	3	3	3	3	3	3
-10	3	3	3	3	3	3	3	3	3	3

Table 4.36: Number of identified conflicting constraints with respect to tol_{rank}(column) and tol_{rc}(row) on the teapot

From the three tables, we can see that both of the two tolerances influence the number of redundant constraints (table 4.37) and numerical over-constraints (table 4.35). The number of conflicting constraints, however, remains the same since the differences of conflicting constraints in figure 4.8 are larger than the range of specified tolerances. This case is specific since these differences are added intentionally large at the beginning to test the algorithm. In practical computation, one has to pay attention to the specification of the two thresholds, which, as demonstrated in this example, affects the detection results.

Actually, the two thresholds are to be consistent with respect to the accuracy of the adopted CAD modeler. So, if the CAD modeler is using an accuracy of 1e-4, then our algorithm should be set up with this value for the two thresholds.

4.5 Conclusion

In this work, the over-constraints detection and resolution process have been described and analyzed with results on both academic and industrial examples. More specially, in the Double-Banana geometry, we shown that

our approach generates a solution that is much closer to the initial design intent than the one of Moinet. In the teapot geometry, we concluded that linearizion of non-linear systems is not reliable since it will induce more over-constraints than what to test really. In the glass geometry, we have been illustrated that the selection of over-constraints set influences the deviation from initial intent. Finally, we shown the efficiency of the decomposition method used in our algorithm and demonstrated that the specification of tolerances would affect the detection result.

Conclusion and perspectives

The general objective of this work was the detection and treatment of geometric over-constraints during the manipulation of free form surfaces. It results in an algorithm detecting geometric over-constraints (redundant and conflicting constraints) as well as finding the corresponding spanning groups. The detection of the over-constraints is at the level of equations but the treatment is at the level of geometries, which enables the manipulation of geometric constraints well suited to engineering design. In this way, the treatment tries to maintain the design intent.

This work has been decomposed in two main categories: the definition of **geometric over-constraints, methods and tools for the detection of geometric over-constraints** (Chapter 2), the selection and integration of these methods to give rise to the algorithms for identifying **geometric over-constraints of free form configurations as well as high level manipulations of these over-constraints** (Chapter 3). The proposed approach allows the detection and treatment of redundant and conflicting constraints. The proposed concepts and algorithms have been implemented and tested on both academic and industrial examples (Chapter 4).

Several practical conclusions have already introduced various perspectives relative to the developed modules. This final section of the manuscript mostly focuses on other **perspectives** which have not been discussed before.

Basic definitions, detection methods and tools...

The chapter 2 of this document describes a set of basic definitions of geometric over-constraints, methods and tools that are able to detect geometric over-constraints. Since the geometric over-constraints can be defined structurally or numerically, the corresponding detection methods and tools are classified into structural detection methods and numerical detection methods. However, these structural definitions and detection methods are mainly

for systems containing only Euler geometries. For systems made from free form geometries, there is no structural definitions i.e. definitions based on DoF counting. As a result, we adopt the numerical definitions of geometric over-constraints for free form geometries, which would require a constraint system represented in an equation form and methods working at the level of equations. To select methods that are useful for identifying geometric over-constraints of free form configurations, several cases are used, where methods are tested and compared with respect to a specified criterion.

One perspective can stem from this part of the work. Formal definitions of geometric over-constraints in terms of free-form geometry should be given while considering its local support property. Also, structural representation of free form configurations as well as the corresponding decomposition methods should be discussed. The decomposition should taken into account the local support property as well.

Over-constraints detection and resolution in geometric equation systems...

Free form configurations can be represented with a set of polynomial equations and the problem is thus transformed into finding numerical over-constraints from the equation set. More specially, since numerical over-constraints are either redundant or conflicting, a set of consistent and inconsistent equations are to be detected. To find them, the chapter 3 describes a tool which is a combination of the structural decompositions and numerical analysis methods. In addition, our approach is able to provide different sets depending on the selected structural decomposition, and proposed criteria to compare them and assist the user in choosing the constraints he/she wants to remove. Moreover, the spanning groups of the over-constraints are detected and help users to quickly locate which set of constraints generates these over-constraints. This makes the debugging process easier. The kernel of the proposed approach works on equations and variables, but the decision is taken by considering the geometric constraints specified by the designer at the geometric level. In chapter 4, the over-constraints detection and resolution process have been described and analyzed with results on both academic and industrial examples. Our approach uses a general DoF-based constriction check enhanced by a WCM-based validation in a recursive assembly way, which allows for interleave decomposition and recombination of system of equations. As shown in the testing cases, it is better than any existing detection method with respect to generality and reliability. Here, the generality refers to the scale of types of geometries and constraints while

the latter refers to the detected over-constraints satisfying our definition of redundant and conflicting constraints.

A number of perspectives stem from this part of work:

- We have restricted the variables to control points in this manuscript. Parameters like degrees of curves/surfaces, weights of the NURBS can also be set as variables, leaving more freedom to the users when manipulating free form geometries. Generally speaking, our algorithm can analyze such cases as well, since it is initially designed for analyzing system of polynomial equations. The generality of our algorithm can be tested when more parameters other than coordinates of control points are set as variables. However, our algorithm cannot be directly used for cases where knots in knots sequences are set as unknowns. This is due to the fact that in this case, computing positions, derivatives on curves/surfaces uses recursive approach. No equations can be generated without knowing the values of knots sequences. Without equations, our approach cannot be set up.

- This give rise to second perspective, that is, develop tools to detect over-constrained configurations when no equations (black box constraints) are available. The tools should detect inconsistencies and give feedbacks to the experts on how to modify them.

- An automation of the process should assist the designer in selecting the set of over-constraints that less deviate from his/her original design intent. As it is, the designer has access to three main criteria (dg, df, $cond$) which can be difficult to analyze for a non-expert. Thus, higher-level criteria should be imagined on top of those ones.

- The approach can be made interactive, i.e. allowing the designer to select between the different conflicting sets along the process, or even modify the faulty constraints.

- Complete solver like interval analysis can be used to solve system of equations. The current solver in our detection framework is based on Levenberg–Marquardt (LM) algorithm, which converges to local minima rather than global one when solving system of equations. Interval analysis is an approach to putting bounds on rounding errors and measurement errors in mathematical computation and thus developing numerical methods that yield reliable results. The solver can be incorporated to our detection framework to analyze academic/industrial cases and compare results with the ones of LM solver.

- Find the minimum number of spanning groups of an over-constraint. Our method enables to find one of spanning groups but not necessarily minimum one. This can be extended in the future. Presenting a user the minimum number of spanning group of an over-constraint will help him/her avoid considering too many constraints when debugging an over-constrained system.

Towards advanced detection of solvable configurations generating poor quality solutions...

It is planned to extend this work so that it can be used to detect and explain geometric configurations which, even when solvable, result in poor quality designs. In the context of free form surfaces deformation, configurations with poor quality are of various types: surfaces with ridges, saddles, troughs, domes, saddle ridges etc. Using artificial intelligence techniques, the rules linking the input parameters to the generation of bad shapes could be obtained by learning carefully selected bad quality examples. Once these rules are obtained, they can be used on a new case to a priori estimate the impact of input parameters without having to perform it.

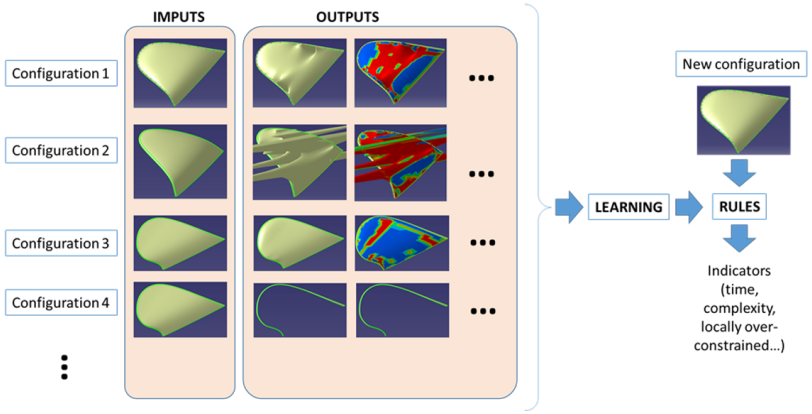


Figure 4.33: Applying machine learning to classify solvable configurations.

Bibliography

Ait-Aoudia, Samy, Roland Jegou, and Dominique Michelucci (2014). "Reduction of constraint systems". In: *arXiv preprint arXiv:1405.6131*.

Bang-Jensen, Jørgen and Gregory Z Gutin (2008). *Digraphs: theory, algorithms and applications*. Springer Science & Business Media.

Bartoň, Michael, Gershon Elber, and Iddo Hanniel (2011). "Topologically guaranteed univariate solutions of underconstrained polynomial systems via no-loop and single-component tests". In: *Computer-Aided Design* 43.8, pp. 1035–1044.

Bettig, Bernhard and Christoph M Hoffmann (2011). "Geometric constraint solving in parametric computer-aided design". In: *Journal of computing and information science in engineering* 11.2, p. 021001.

Boehm, Wolfgang and Hartmut Prautzsch (1985). "The insertion algorithm". In: *Computer-Aided Design* 17.2, pp. 58–59.

Bose, NK (1995). "Gröbner bases: An algorithmic method in polynomial ideal theory". In: *Multidimensional Systems Theory and Applications*. Springer, pp. 89–127.

Bunus, Peter and Peter Fritzson (2002a). "A debugging scheme for declarative equation based modeling languages". In: *International Symposium on Practical Aspects of Declarative Languages*. Springer, pp. 280–298.

— (2002b). "Methods for structural analysis and debugging of Modelica models". In: *Proceedings of the 2nd International Modelica Conference*. Vol. 10, pp. 157–165.

Cheutet, Vincent, Marc Daniel, Stefanie Hahmann, Raphaël La Greca, Jean-Claude Léon, Robert Maculet, David Ménegaux, and Basile Sauvage (2007). "Constraint modeling for curves and surfaces in CAGD: A survey". In: *International Journal of Shape Modeling* 13.02, pp. 159–199.

Chou, S.C. (1988a). *Mechanical Geometry Theorem Proving*. Mathematics and Its Applications. Springer. ISBN: 9789027726506.

Chou, Shang-Ching (1988b). "An introduction to Wu's method for mechanical theorem proving in geometry". In: *Journal of Automated Reasoning* 4.3, pp. 237–267.

Chou, Shang-Ching and Xiao-Shan Gao (1990). "Ritt-Wu's decomposition algorithm and geometry theorem proving". In: *10th International Conference on Automated Deduction*. Springer, pp. 207–220.

Cox, David, John Little, and Donal O'shea (1992). *Ideals, varieties, and algorithms*. Vol. 3. Springer.

Cox, David A, John Little, and Donal O'Shea (2015). "Groebner Bases". In: *Ideals, varieties, and algorithms*. Springer, pp. 49–119.

Coxeter, Harold Scott Macdonald and Samuel L Greitzer (1967). *Geometry revisited*. Vol. 19. Maa.

Diestel, R. (2006). *Graph Theory*. Electronic library of mathematics. Springer. ISBN: 9783540261834.

Dongarra, J.J. and S.A.G. Supercomputing (1990). *Proceedings of the Fourth SIAM Conference on Parallel Processing for Scientific Computing*. Proceedings in Applied Mathematics Series. Society for Industrial and Applied Mathematics. ISBN: 9780898712629.

Dubé, Thomas W (1990). "The structure of polynomial ideals and Gröbner bases". In: *SIAM Journal on Computing* 19.4, pp. 750–773.

Dulmage, Andrew L and Nathan S Mendelsohn (1958). "Coverings of bipartite graphs". In: *Canadian Journal of Mathematics* 10.4, pp. 516–534.

Elber, Gershon (2000). *Linearizing the area and volume constraints*. Tech. rep. Technical Report.

— (2001). "Multiresolution curve editing with linear constraints". In: *Proceedings of the sixth ACM symposium on Solid modeling and applications*. ACM, pp. 109–119.

Falcidieno, Bianca, Franca Giannini, Jean-Claude Léon, and Jean-Philippe Pernot (2014). "Processing free form objects within a Product Development Process framework". In: *Advances in Computers and Information in Engineering Research*. Vol. 1. ASME-Press, pp. 317–344.

Farin, Gerald E, Josef Hoschek, and Myung-Soo Kim (2002). *Handbook of computer aided geometric design*. Elsevier.

Fontana, Marzia, Franca Giannini, and Maria Meirana (2000). "Free form features for aesthetic design". In: *International Journal of Shape Modeling* 6.02, pp. 273–302.

Ford Jr, LR and DR Fulkerson (2009). "Maximal flow through a network". In: *Classic papers in combinatorics*. Springer, pp. 243–248.

Foufou, Sebti and Dominique Michelucci (2012). "Interrogating witnesses for geometric constraint solving". In: *Information and Computation* 216, pp. 24–38.

Foufou, Sebti, Dominique Michelucci, and Jean-Paul Jurzak (2005). "Numerical decomposition of geometric constraints". In: *Proceedings of the*

2005 ACM symposium on Solid and physical modeling. ACM, pp. 143–151.

Fudos, Ioannis and Christoph M Hoffmann (1997). "A graph-constructive approach to solving systems of geometric constraints". In: *ACM Transactions on Graphics (TOG)* 16.2, pp. 179–216.

Gao, Xiao-Shan and Shang-Ching Chou (1998). "Solving geometric constraint systems. II. A symbolic approach and decision of Rc-constructibility". In: *Computer-Aided Design* 30.2, pp. 115–122.

Ge, Jian-Xin, Shang-Ching Chou, and Xiao-Shan Gao (1999). "Geometric constraint satisfaction using optimization methods". In: *Computer-Aided Design* 31.14, pp. 867–879.

Golub, G.H. and C.F. Van Loan (2013). *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press. ISBN: 9781421407944.

Gonzalez-Ochoa, Carlos, Scott McCammon, and Jörg Peters (1998). "Computing moments of objects enclosed by piecewise polynomial surfaces". In: *ACM Transactions on Graphics (TOG)* 17.3, pp. 143–157.

Gouaty, Gilles, Lincong Fang, Dominique Michelucci, Marc Daniel, Jean-Philippe Pernot, Romain Raffin, Sandrine Lanquetin, and Marc Neveu (2016). "Variational geometric modeling with black box constraints and DAGs". In: *Computer-Aided Design* 75, pp. 1–12.

Graver, J.E., B. Servatius, and H. Servatius. *Combinatorial Rigidity*. Graduate studies in mathematics. American Mathematical Soc.

Guillet, Stéphane (1999). "Modification et construction de formes gauches soumises à des contraintes de conception". PhD thesis.

Hahmann, Stefanie, Basile Sauvage, and Georges-Pierre Bonneau (2005). "Area preserving deformation of multiresolution curves". In: *Computer aided geometric design* 22.4, pp. 349–367.

Haug, Edward J (1989). *Computer aided kinematics and dynamics of mechanical systems*. Vol. 1. Allyn and Bacon Boston.

Hoffman, Christoph M, Andrew Lomonosov, and Meera Sitharam (2001). "Decomposition plans for geometric constraint systems, Part I: Performance measures for CAD". In: *Journal of Symbolic Computation* 31.4, pp. 367–408.

Hoffmann, Christoph M. (1989). *Geometric and Solid Modeling: An Introduction*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN: 1-55860-067-1.

Hoffmann, Christoph M and Robert Juan (1992). "Erep: An editable, high-level representation for geometric design and analysis". In: *Selected and Expanded Papers from the IFIP TC5/WG5. 2 Working Conference on*

Geometric Modeling for Product Realization. North-Holland Publishing Co., pp. 129–164.

Hoffmann, Christoph M, Andrew Lomonosov, and Meera Sitharam (1997). "Finding solvable subsets of constraint graphs". In: *International Conference on Principles and Practice of Constraint Programming*. Springer, pp. 463–477.

— (1998). "Geometric constraint decomposition". In: *Geometric constraint solving and applications*. Springer, pp. 170–195.

Hoffmann, Christoph M, Meera Sitharam, and Bo Yuan (2004). "Making constraint solvers more usable: overconstraint problem". In: *Computer-Aided Design* 36.4, pp. 377–399.

Hu, Hao, Mathias Kleiner, and Jean-Philippe Pernot (2017). "Over-constraints detection and resolution in geometric equation systems". In: *Computer-Aided Design*.

Jermann, Christophe, Bertrand Neveu, and Gilles Trombettoni (2003). "Algorithms for identifying rigid subsystems in geometric constraint systems". In: *IJCAI*. Vol. 3, pp. 233–238.

Jermann, Christophe, Gilles Trombettoni, Bertrand Neveu, and Pascal Mathis (2006). "Decomposition of geometric constraint systems: a survey". In: *International Journal of Computational Geometry & Applications* 16.05n06, pp. 379–414.

Kincaid, David Ronald and Elliott Ward Cheney (2002). *Numerical analysis: mathematics of scientific computing*. Vol. 2. American Mathematical Soc.

Kondo, Kunio (1992). "Algebraic method for manipulation of dimensional relationships in geometric models". In: *Computer-Aided Design* 24.3, pp. 141–147.

Krantz, S.G. and H.R. Parks (2012). *The Implicit Function Theorem: History, Theory, and Applications*. Modern Birkhäuser Classics. Springer New York. ISBN: 9781461459811.

Kubicki, Arnaud, Dominique Michelucci, and Sebti Foufou (2014). "Witness computation for solving geometric constraint systems". In: *Science and Information Conference (SAI), 2014*. IEEE, pp. 759–770.

Laman, Gerard (1970). "On graphs and rigidity of plane skeletal structures". In: *Journal of Engineering mathematics* 4.4, pp. 331–340.

Lamure, Hervé and Dominique Michelucci (1998). "Qualitative study of geometric constraints". In: *Geometric Constraint Solving and Applications*. Springer, pp. 234–258.

Lasseter, John (1987). "Principles of traditional animation applied to 3D computer animation". In: *ACM Siggraph Computer Graphics*. Vol. 21. 4. ACM, pp. 35–44.

Latham, Richard S and Alan E Middleditch (1996). "Connectivity analysis: a tool for processing geometric constraints". In: *Computer-Aided Design* 28.11, pp. 917–928.

Leiserson, Charles E and Tao B Schardl (2010). "A work-efficient parallel breadth-first search algorithm (or how to cope with the nondeterminism of reducers)". In: *Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures*. ACM, pp. 303–314.

Lesage, David (2002). "Un modèle dynamique de spécifications d'ingénierie basé sur une approche de géométrie variationnelle". PhD thesis. Grenoble, INPG.

Light, Robert A and David C Gossard (1983). "Variational geometry: a new method for modifying part geometry for finite element analysis". In: *Computers & Structures* 17.5-6, pp. 903–909.

Lin, Vincent C, David C Gossard, and Robert A Light (1981). "Variational geometry in computer-aided design". In: *ACM SIGGRAPH Computer Graphics* 15.3, pp. 171–177.

Maculet, Robert and Marc Daniel (2004). "Conception, modélisation géométrique et contraintes en CAO: une synthèse." In: *Revue d'intelligence artificielle* 18.5-6, pp. 619–645.

Michalik, P and BD Brüderlin (2004). "Constraint-based design of B-spline surfaces from curves". In: *Proceedings of the ninth ACM symposium on Solid modeling and applications*. Eurographics Association, pp. 213–223.

Michelucci, Dominique and Sebti Foufou (2006a). "Detecting all dependences in systems of geometric constraints using the witness method". In: *International Workshop on Automated Deduction in Geometry*. Springer, pp. 98–112.

— (2006b). "Geometric constraint solving: The witness configuration method". In: *Computer-Aided Design* 38.4, pp. 284–299.

Michelucci, Dominique, Sebti Foufou, Loic Lamarque, and Pascal Schreck (2006). "Geometric constraints solving: some tracks". In: *Proceedings of the 2006 ACM symposium on Solid and physical modeling*. ACM, pp. 185–196.

Michelucci, Dominique, Pascal Schreck, Simon EB Thierry, Christoph Fünfzig, and Jean-David Génevaux (2010). "Using the witness method to detect rigid subsystems of geometric constraints in CAD". In: *Proceedings of the 14th ACM Symposium on Solid and Physical Modeling*. ACM, pp. 91–100.

Moinet, Mireille, Guillaume Mandil, and Philippe Serre (2014). "Defining tools to address over-constrained geometric problems in Computer Aided Design". In: *Computer-Aided Design* 48, pp. 42–52.

Okunev, Pavel and Charles R Johnson (2005). "Necessary and sufficient conditions for existence of the LU factorization of an arbitrary matrix". In: *arXiv preprint math/0506382*.

Owen, John C (1991). "Algebraic solution for geometry from dimensional constraints". In: *Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*. ACM, pp. 397–407.

— (1996). "Constraints on simple geometry in two and three dimensions". In: *International Journal of Computational Geometry & Applications* 6.04, pp. 421–434.

Oxley, J.G. (2006). *Matroid Theory*. Oxford graduate texts in mathematics. Oxford University Press. ISBN: 9780199202508.

Pernot, J-P, Stephane Guillet, J-C Léon, Bianca Falcidieno, and Franca Giannini (2004). "Multi-minimisations for shape control of fully free-form deformation features (/spl delta/-F/sup 4/)". In: *Shape Modeling Applications, 2004. Proceedings.* IEEE, pp. 53–62.

Pernot, J-P, Bianca Falcidieno, Franca Giannini, and J-C Léon (2008). "Incorporating free-form features in aesthetic and engineering product design: State-of-the-art report". In: *Computers in Industry* 59.6, pp. 626–637.

Pernot, Jean-Philippe (2004). "Fully free form deformation features for aesthetic and engineering designs". PhD thesis. Grenoble, INPG.

Pernot, Jean-Philippe, Qian Qiao, and Philippe Veron (2007). "Constraints Automatic Relaxation to Design Products with Fully Free Form Features". In: *Advances in Integrated Design and Manufacturing in Mechanical Engineering II*, pp. 145–160.

Pernot, Jean-philippe, Bianca Falcidieno, Franca Giannini, and Jean-claude Léon (2005). "Fully free-form deformation features for aesthetic shape design". In: *Journal of Engineering Design* 16.2, pp. 115–133.

Piegl, L. and W. Tiller (1996). *The NURBS Book*. Monographs in Visual Communication. Springer Berlin Heidelberg. ISBN: 9783540615453.

Podgorelec, David, Borut Žalik, and Vid Domiter (2008). "Dealing with redundancy and inconsistency in constructive geometric constraint solving". In: *Advances in Engineering Software* 39.9, pp. 770–786.

Pothen, Alex and Chin-Ju Fan (1990). "Computing the block triangular form of a sparse matrix". In: *ACM Transactions on Mathematical Software (TOMS)* 16.4, pp. 303–324.

Sauvage, Basile, Stefanie Hahmann, and Georges-Pierre Bonneau (2004). "Length preserving multiresolution editing of curves". In: *Computing* 72.1, pp. 161–170.

Serrano, David (1987). "Constraint management in conceptual design". PhD thesis. Massachusetts Institute of Technology.

Serrano, David (1991). "Automatic dimensioning in design for manufacturing". In: *Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*. ACM, pp. 379–386.

Sitharam, Meera, Jörg Peters, and Yong Zhou (2004). "Solving minimal, wellconstrained, 3d geometric constraint systems: combinatorial optimization of algebraic complexity". In: *Automated deduction in Geometry (ADG)*.

Sitharam, Meera and Yong Zhou (2004). "A tractable, approximate, combinatorial 3d rigidity characterization". In: *Fifth Automated Deduction in Geometry (ADG)*.

Sridhar, Natarajan, Rajiv Agrawal, and Gary L Kinzel (1996). "Algorithms for the structural diagnosis and decomposition of sparse, underconstrained design systems". In: *Computer-Aided Design* 28.4, pp. 237–249.

Strang, G. (2006). *Linear Algebra and Its Applications*. Thomson, Brooks/Cole. ISBN: 9780030105678.

Tarjan, Robert (1972). "Depth-first search and linear graph algorithms". In: *SIAM journal on computing* 1.2, pp. 146–160.

Thierry, Simon EB, Pascal Schreck, Dominique Michelucci, Christoph Fünfzig, and Jean-David Génevaux (2011). "Extensions of the witness method to characterize under-, over-and well-constrained geometric constraint systems". In: *Computer-Aided Design* 43.10, pp. 1234–1249.

Thulasiraman, Krishnaiyan and Madisetti NS Swamy (2011). *Graphs: theory and algorithms*. John Wiley & Sons.

Wu, Wen-tsün (2012). *Mechanical theorem proving in geometries: Basic principles*. Springer Science & Business Media.

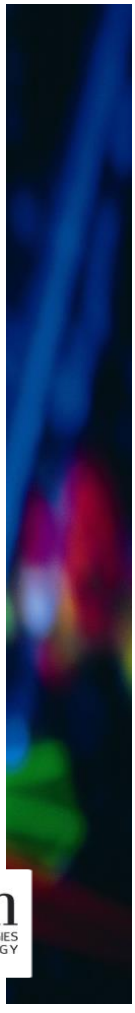
Xiaobo, Peng, Chen Liping, Zhou Fanli, and Zhou Ji (2002). "Singularity analysis of geometric constraint systems". In: *J. Comput. Sci. & Technol* 17.3, pp. 314–323.

DÉTECTION ET TRAITEMENT DE CONFIGURATIONS INCONTINENTES OU LOCALEMENT CONTRAINTES PENDANT LA MANIPULATION DE MODÈLES GÉOMÉTRIQUES 3D EN SURFACES DE FORME LIBRE

RESUME: Les modeleurs de CAO actuels doivent intégrer des fonctionnalités de plus en plus avancées pour la modélisation de produits de haute qualité définis par des formes complexes. Il est notamment nécessaire de développer des outils d'aide à la décision pour aider les concepteurs lors de la phase de modélisation géométrique. En fait, la forme finale d'un produit résulte souvent de la satisfaction d'exigences esquissées au cours d'un processus de conception itératif. Les exigences peuvent être considérées comme des contraintes à satisfaire. Pour définir un objet de forme libre, les concepteurs doivent spécifier les contraintes géométriques que l'objet doit satisfaire. La plupart du temps, ces contraintes génèrent un ensemble d'équations linéaires et non-linéaires reliant des variables dont les valeurs doivent être trouvées. Pendant le processus de modélisation, les concepteurs peuvent exprimer involontairement plusieurs fois les mêmes exigences en utilisant différentes contraintes conduisant ainsi à des équations redondantes. Ils peuvent également générer involontairement des équations contradictoires produisant des configurations insatisfiable. En outre, en raison de la propriété de support locale des NURBS, les équations peuvent ne pas s'étendre à toutes les variables, entraînant ainsi des sous-parties localement potentiellement surcontraintes.

Cette thèse propose une approche originale d'aide à la décision pour aborder les configurations géométriques sur-contraintes durant les phases de conception assistée par ordinateur. Elle se concentre particulièrement sur la détection et la résolution de contraintes redondantes et conflictuelles lors de la déformation de surfaces libres à base de carreaux NURBS. À partir d'une série de décompositions structurelles couplées à des analyses numériques, l'approche proposée traite à la fois les contraintes linéaires et non-linéaires. Les décompositions structurelles sont particulièrement efficaces en raison de la propriété de support local des NURBS. Puisque le résultat de ce processus de détection n'est pas unique, plusieurs critères sont introduits pour inciter le concepteur à identifier les contraintes à supprimer afin de minimiser l'impact sur son intention de conception initiale. Ainsi, même si le noyau de l'algorithme travaille sur des équations et des variables, la décision est prise en considérant les contraintes géométriques spécifiées par l'utilisateur.

Mots clés : Processus de développement de produit, déformation de surface de forme libre, équations linéaires et non linéaires, sous-parties localement sur-contraintes, contraintes redondantes et conflictuelles, décomposition structurelle, analyse numérique, aide à la décision, intention de conception



ARTS ET MÉTIERS ParisTech

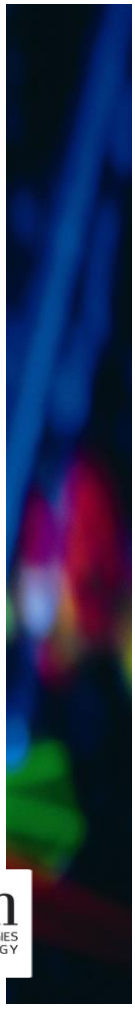
ParisTech INSTITUT DES SCIENCES ET TECHNOLOGIES PARIS INSTITUTE OF TECHNOLOGY

DETECTION AND TREATMENT OF INCONSISTENT OR LOCALLY OVER-CONSTRAINED CONFIGURATIONS DURING THE MANIPULATION OF 3D GEOMETRIC MODELS MADE OF FREE-FORM SURFACES

ABSTRACT: Today's CAD modelers need to incorporate more and more advanced functionalities for the modeling of high-quality products defined by complex shapes. There is notably a need for developing decision support tools to help designers during the free-form geometric modeling phase of the Product Design Process (PDP). Actually, the final shape of a product often results from the satisfaction of requirements sketched during an incremental design process. The requirements can be seen as constraints to be satisfied. To shape a free-form object, designers have to specify the geometric constraints the object has to satisfy. Most of the time, those constraints will generate a set of linear and non-linear equations linking variables whose values have to be found. During the modeling process, designers may express involuntarily several times the same requirements using different constraints thus leading to redundant equations, or they can involuntarily generate conflicting equations resulting in unsatisfiable configurations. Additionally, due to the local support property of NURBS, equations may not span on all variables, thus resulting in locally over-constrained subparts.

This work proposes an original decision-support approach to address over-constrained geometric configurations in Computer-Aided Design. It focuses particularly on the detection and resolution of redundant and conflicting constraints when deforming free-form surfaces made of NURBS patches. Based on a series of structural decompositions coupled with numerical analyses, the proposed approach handles both linear and non-linear constraints. The structural decompositions are particularly efficient because of the local support property of NURBS. Since the result of this detection process is not unique, several criteria are introduced to drive the designer in identifying which constraints should be removed to minimize the impact on his/her original design intent. Thus, even if the kernel of the algorithm works on equations and variables, the decision is taken by considering the user-specified geometric constraints.

Keywords : Product Development Process, free-form surface deformation, linear and non-linear equations, locally over-constrained subparts, redundant and conflicting constraints, structural decomposition, numerical analysis, decision support, design intent



ARTS ET MÉTIERS ParisTech

ParisTech INSTITUT DES SCIENCES ET TECHNOLOGIES PARIS INSTITUTE OF TECHNOLOGY