HAL
open science

Geometric Constraint Solver

Adel Moussaoui

To cite this version:

Adel Moussaoui. Geometric Constraint Solver. Computational Geometry [cs.CG]. Ecole nationale Supérieure d'Informatique (ex I.N.I), Alger, 2016. English. NNT : . tel-01402691

HAL Id: tel-01402691

https://hal.science/tel-01402691v1

Submitted on 25 Nov 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieure et de la Recherche Scientifique
Ecole Nationale Supérieure en Informatique

Thèse

En vue de l'otention du titre de:

Doctorat ès sciences

Specialité:

Informatique

Par:

MOUSSAOUI Adel

Dirigé par: Pr. AIT AOUDIA Samy

Geometric Constraint Solver

Soutenue le 24 novembre 2016 devant le jury:

M.	BALLA Amar	Prof.	ESI, Alger	President
M.	HADDADOU Hamid	MCA	ESI, Alger	Examinateur
M.	DJEDI Noureddine	Prof.	Université de Biskra	Examinateur
M.	MICHELUCCI Dominique	Prof.	Université de Dijon, France	Examinateur
M.	AIT AOUDIA Samy	Prof.	ESI, Alger	Directeur de thèse

Abstract

Doctor of science

Geometric Constraint Solver

by Adel MOUSSAOUI

A geometric constraint system consists of a finite set of geometric elements, such as points, lines, and circles, along with relationships of different types such as distance, angle, incidence and parallelism. This problem is central to many applications, such as computer-aided design, molecular modelling and recently localization in wireless sensor networks. Solving a geometric constraint system consists of finding real coordinates of geometric elements in the Euclidean space. In 2-dimensional geometric constraint solving, graph-based techniques are a dominant approach, particularly in the computer-aided design context. To speed up the resolution process, these methods transform the geometric problem into a graph, which is decomposed into small subgraphs. Each one is solved, separately, and the final solution is obtained by recomposing the solved subgraphs. However, most of the previous research on graph-based approaches has only focused on the decomposition without any attention on what will be decomposed: the geometric constraint graph. Major proposed algorithms are discussed or compared theoretically, without presenting any tests on graphs instances with different structural properties, representing several cases of difficulties. Why? because as far as we know, there is no known algorithm for the creation of non-decomposable graphs or graphs with interesting structural properties that best highlight the efficiency of any algorithm. Our contribution is the design of a simple, but efficient random 2D geometric constraint graph generator. It can be used to make benchmarks for consistent tests, or to observe the behaviour of geometric constraints solving algorithms. It produces problem instances with various sizes and structural properties, covering different cases of complexity. Our design is based on the problem classification reported in the literature. We proved that our proposed generator is complete, customizable, simple and efficient. It has been validated experimentally and some of its properties have been theoretically proved.[1]

Résumé

Docteur ès sciences

Solveur de systèmes de contraintes géometriques

par Adel MOUSSAOUI

Un système de contraintes géométriques est constitué d'un ensemble fini d'éléments géométriques (points, lignes, cercles ...) et de relations géométriques tels que la distance, le parallélisme, l'angle, l'incidence, etc. Ce problème est central dans de nombreuses applications, en particulier la CAO. Résoudre un système de contraintes géométriques consiste à trouver des coordonnées réelles des éléments géométriques dans l'espace Euclidien de telle sorte que toutes les contraintes du système soient satisfaites. Dans l'espace 2D, les approches basées graphes sont dominantes. Afin d'accélérer la résolution, ces méthodes transforment le système de contraintes en un graphe qui sera ensuite analysé puis décomposé en petits sous-graphes. Chaque sous-graphe est résolu, séparément, la solution finale est obtenue en recomposant les sous-graphes résolus. Cependant, la plupart des algorithmes proposés sont analysés ou comparés théoriquement sans présenter des tests sur des cas de graphes de contraintes avec différentes propriétés structurales, représentant plusieurs cas de difficultés. Cela est dû à l'absence de méthodes connues pour la création de graphes de contraintes non décomposables ou ayant des propriétés structurelles qui illustrent mieux l'efficacité des algorithmes. Notre contribution est la conception d'un générateur aléatoire de graphe de contraintes qui est simple et efficace. Il peut être utilisé pour des analyses de performances, permettant des tests cohérents, ou bien pour observer le comportement des solveurs de systèmes de contraintes géométriques. Notre générateur produit des instances de problèmes avec différentes tailles et propriétés structurelles. Notre conception est basée sur la classification des problèmes rapportés dans la littérature. Nous avons prouvé que notre générateur est complet, paramétrable, simple et efficace. Il a été validé expérimentalement et certaines de ses propriétés ont été théoriquement prouvées.

المدرسة الوطنية العليا للإعلام الآلي

ملخص

دكتوراه علوم

موساوي عادل

يتكون النظام الهندسي المقيد من مجموعة محدودة من العناصر الهندسية، مثل النقاط، المستقيمات، الدوائر وغيرها، تربطها علاقات مختلفة كالبعد، الزاوية و التوازي. لهذه المسألة أهمية أساسية في العديد من التطبيقات، كالتصميم بمساعدة الحاسوب، النمذجة الهندسية ومؤخرا تحديد المكان في شبكات الاستشعار اللاسلكية. يتمثل حل النظام الهندسي المقيد في تعيين الإحداثيات الحقيقية للعناصر الهندسية في الفضاء الإقليدي. التقنيات التي تعتمد على المخطط البياني هي النهج السائد في الهندسة المستوية ولا سيما في سياق التصميم بمساعدة الحاسوب. لتسريع عملية الحل، عادة ما يتم تفكيك المخطط البياني الى مخططات جزئية، يتم بعد ذلك حل كل مخطط جزئي على حدى ليتم الحصول على الحل النهائي بواسطة جبر المخططات البيانية الفرعية. تركز معظم الأبحاث المنشورة و خاصة تلك التي تعتمد على المخطط البياني على التحليل النضري البحت مع بعض الأمثلة البسيطة. دائما ما تقارن الخوارزميات المقترحة مقارنة نظرية دون تقديم أي اختبارات على مخططات بيانية متعددة الخصائص الهيكلية مع درجات تعقيد مختلفة. المشكل يكمن في عدم توفر قاعدة بيانات لمسائل القيد الهندسي متدرجة التعقيد، و بقدر ما نعلم، لا توجد لحد الآن أي خوارزمية معروفة لإنشاء مخططات بيانية عشوائية ذات درجات تعقيد أو بنية هيكلية محددة. الهدف من هذه الرسالة هو تصميم مولد مخططات بيانية تستعمل لاختبار كفاءة أي خوارزمية في حل القيود الهندسية. لقد قمنا بتصميم مولد عشوائي بسيط، ولكنه فعال. يمكن استخدامه لجعل معايير الاختبارات اكثر اتساقا، أو مراقبة سلوك الخوارزميات مع قيود هندسية متباينة. يمكن لهذا المولد إنتاج مسائل مختلفة الأحجام والخصائص الهيكلية مع درجات تعقيد مختلفة . استندنا في تصميم المولد على تصنيفات المشكلة في الأعمال المنشورة سابقا. لقد أثبتنا أن المولد المقترح كامل، قابل للتخصيص، بسيط وفعال. تم التحقق من صحة ذلك بالتجربة و بالبرهان على بعض خصائصه نظريا.

Acknowledgements

In any case, I am indebted to many people for making the time working on my doctoral thesis an unforgettable experience.

I would like to express my deepest gratitude to my advisor, Pr. Samy AIT AOUDIA, for his excellent guidance, caring, patience, motivation, and providing me with a good atmosphere for doing research. I could not have imagined having a better advisor for my thesis.

I am deeply grateful to all the members of the examination committee for agreeing to read the manuscript and accepting to participate in the defence of this thesis: Pr. BALLA Amar, the committee president, and Pr. Dominique MICHELUCCI, Pr. Noureddine DJEDI and Dr. Hamid HADDADOU. Their suggestions will be taken into account and will significantly improve the final version.

My thanks also go to two anonymous experts who have examined the manuscript. I also express my gratitude to all of the ESI administration staff for their help and support.

Last but not the least, I express my very profound gratitude to my parents and my wife and all the family for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. I also thank my dearest friends for their understanding and encouragement.

Contents

Abstract i

Résumé i

Acknowledgements iv

Contents v

List of Figures viii

List of Tables xi

List of Algorithms xii

Abbreviations xiii

1 Introduction 1
- 1.1 Preface . . . 1
- 1.2 Objectives and contributions . . . 4
- 1.3 Thesis organization . . . 6

2 Background 8
- 2.1 Geometric constraints solving: problem definitions . . . 9
- 2.2 Modelling a GCS by a system of equations . . . 11
- 2.3 Graph theory . . . 13
  - Graph: . . . 13
  - Incidence, adjacency and degree: . . . 14
  - Cardinality and size: . . . 14
  - Subgraph: . . . 14
  - Path and connectivity: . . . 14

Walk, trail and circuit: 14
Acyclic graph and spanning trees: 15
Planar graph: 15
Bipartite graph: 15
Graph matching: 15
Maximum and perfect matching: 15
Alternating and augmenting path: 16
Dulmage–Mendelsohn decomposition: 16
2.4 The importance of GCS decomposition in CAD 16
2.5 NP-Hardness of finding real solution in 2D 19
2.6 Rigidity theory and Laman theorem 20
2.6.1 Laman condition 21
2.6.2 Rigidity testing in 2D: The pebble game 23
2.7 Generalization of Laman's theorem to other types of constraints 27
2.8 Combinatorial characterization of the 3D case 29
2.9 Other application of geometric constraints solving 31
2.9.1 Localisation in Wireless Sensor Networks 31
2.9.2 Molecular Biology 32
2.9.3 Dynamic Geometry 33

3 Major Solving Approaches in CAD 35
3.1 Numerical Solvers 36
3.1.1 Newton-Raphson method 36
3.1.2 Bisection method 37
3.1.3 Homotopy method 39
3.2 Other solving Approaches 40
3.2.1 Degrees of Freedom Analysis Approaches 40
3.2.2 Propagation Approaches 40
3.2.3 Logic-Based Approaches 41
3.2.4 Symbolic Approaches 41
3.3 Conclusion 43

4 2D Graph-based Solvers 44
4.1 Introduction 45
4.2 2D geometric constraint graph and their decomposition 46
4.3 Desirable requirements of a graph-based solver 47
4.4 Constructive Solvers 48
4.4.1 Decomposition analysis method 48
4.4.2 Reduction analysis method 49
4.4.3 Tree Decomposition Method 53
4.5 General Solvers 55
4.5.1 Geometric Constraint Bipartite Network 55

4.5.2 Detection of solvable subgraphs 56
4.5.3 Maximum Matching Approach 59
4.5.4 Some remarks on the MM Algorithm 64
4.5.5 Condensing and Frontier algorithms 65
4.5.5.1 The Condensing Algorithm 66
4.5.5.2 Frontier Algorithm 66
4.5.6 Recursive Skeletonization Algorithm 68
4.6 Conclusion 72

5 Our Contribution: A 2D Geometric Constraint Graph Generator 74
5.1 Introduction 75
Completeness 76
Customizability: 76
Simplicity and efficiency: 76
5.2 Geometric Constraint Graph Decomposition. 77
5.3 Generating well-constrained graphs 78
5.4 Random constraint graph generator 86
5.5 Embedding over- and under-constrained subgraph 95
5.6 Experimental results 95
5.7 Conclusion 97

6 Conclusions and Future Work 100
6.1 Summary and Conclusions 100
6.2 Further works 102

A Some samples of non-decomposable graphs 104

Bibliography 117

List of Figures

1.1 Two well-constrained graphs: (a) solvable by SR-Planners and (b) solvable only by MM-Planners 4

2.1 A well-constrained GCS in 2D. 10
2.2 An over-constrained GCS in 2D. 10
2.3 An under-constrained GCS in 2D. 10
2.4 A piston-crankshaft mechanism. 11
2.5 Geometric constraint system of the piston-crankshaft mechanism 11
2.6 A 2D constrained model. 17
2.7 The corresponding constraints graph. 17
2.8 The extruded model. 18
2.9 The final 3D model. 18
2.10 Flexible, rigid and stressed framework. 21
2.11 Examples of minimally rigid Graphs 22
2.12 Examples of non-minimally rigid graphs. (a) is not rigid. (b), (c) and (d) are over-rigid, i.e., some edges are dependant . . . 23
2.13 Illustration of the pebble game with $|V| = 3$ 25
2.14 Searching for free pebble 27
2.15 Three lines with angle constraint 28
2.16 Pappus Theorem 29
2.17 The double banana, a 3D counter example. 30
2.18 Example of WSN topology. 32
2.19 A Rigidity decomposition of a protein 33
2.20 A GeoGebra screenshot. 34

3.1 A two-dimensional illustration of the bisection method. 39

4.1 Two well-constrained graphs: (a) solvable by SR-Planners and (b) solvable only by MM-Planners 46
4.2 A geometric constraint graph decomposition by Owen's algorithm 50
4.3 Reduction of a 6-cycle of the bipartite graph H that corresponds to a cluster merging. 52
4.4 A geometric constraint graph and its spanning tree. 54
4.5 Set of fundamental circuits of graph 4.4a. 54

4.6 (a) A constraint graph, (b) its bipartite associated network . 57
4.7 (a)A flow for the constraint graph (b) another flow, (c) the initial flow 58
4.8 Maximum matching decomposition algorithm 62
4.9 An example of an incorrect case 64
4.10 Two different MM-decompositions of the same constraint graph. 65
4.11 Constraint graph with vertices of weight 2 and edges of weight 1. The minimal dense subgraph {a, b} can be extended sequentially by the other elements, in alphabetic order. 66
4.12 Condenced Algorithm principle 67
4.13 An example where the CA simplification step fail. 68
4.14 Example of Frontier algorithm 69
4.15 Example of constraint graph clustering and its skeleton 71
4.16 The clusters placement for the skeleton of figure 4.15a. 72

5.1 The first operations of Henneberg construction: HI 79
5.2 The second operations of Henneberg construction: HII 79
5.3 The size of the graph vs degree of decomposability for p = 0, p = 0.5 and p = 1. 81
5.4 The percentage of non decomposable graphs in each set of 20 instances generated, size ranging from 4 to 100, the probability $p = 0$ 81
5.5 The probability of choosing the first operation of Henneberg vs the degree of decomposition. 83
5.6 Three Geometric constraint graphs generated by RH for different values of p 85
5.7 A geometric constraints graph. 87
5.8 Four steps to generate a well-constrained graph solvable only by MM-Planners. 91
5.9 A well-constrained graph of 500 vertices generated by $RRH(500, 0, 50)$. 98

A.1 A non-decomposable graph with 6 nodes. 104
A.2 A non-decomposable graph with 8 nodes. 105
A.3 A non-decomposable graph with 9 nodes. 106
A.4 A non-decomposable graph with 10 nodes. 106
A.5 A non-decomposable graph with 20 nodes. 107
A.6 A non-decomposable graph with 30 nodes. 108
A.7 A non-decomposable graph with 40 nodes. 109
A.8 A non-decomposable graph with 50 nodes. 110
A.9 A non-decomposable graph with 60 nodes. 111
A.10 A non-decomposable graph with 70 nodes. 112
A.11 A non-decomposable graph with 80 nodes. 113

A.12 A non-decomposable graph with 90 nodes. 114
A.13 A non-decomposable graph with 100 nodes. 116

List of Tables

2.1 A set of geometric constraints for the piston-crankshaft mechanism. 12

5.1 Decomposability degree for different values of p : 0, 0.5 and 1. 82

5.2 The degree of decomposition for different values of p and size of the graph 84

5.3 Number of detected subgraphs with $n = 40$ and for different values of p : 0, 0.5 and 1. 86

5.4 The average number of subgraph detected for graphs generated by $RRH(n, 0, m)$ for different values of n and m. 95

5.5 The average number of subgraphs detected for different graphs generated by $RRH(n, 0, m)$ and for different values of n and m. (the expected optimal case: using the procedure RH0 (see algorithm 8)to generate subgraphs by algorithm 9 step 5 ; and for decomposition, using the MM planner modified by pinning all edges, on in turn (section 4.5.4) 96

List of Algorithms

1 Jacobs and Hendrickson algorithm: The pebble game. 26
2 Owen's decomposition algorithm 51
3 Reduction analysis algorithm 52
4 A maximum matching algorithm for GCS decomposition . . . 61
5 The recursive Skeletonization. 70
6 Geometric constraint graph decomposition 71
7 Generate a random well-constrained graph using Henneberg construction according to the probability p. 80
8 Generate a non decomposable well-constrained graph. 83
9 Generation of a random well-constrained graph, according to the two metrics: (1) the size of the largest subgraph, (2) the degree of decomposition. 90

Abbreviations

CAD	**C**omputer **A**ided **D**esign
DOF	**D**egree **O**f **F**reedom
GCG	**G**eometric **C**onstraint **G**raph
GCS	**G**eometric **C**onstraint **S**ystem

In the memory of my grandmother

Chapter 1

Introduction

Contents

1.1 Preface 1

1.2 Objectives and contributions 4

1.3 Thesis organization 6

1.1 Preface

A geometric constraint system (GCS) consists of a finite set of geometric elements along with relationships of different types called constraints. In plane geometry (2D space), we can cite for example, points, lines, and circles, constrained with distance, angle, incidence, and parallelism. In solid geometry (3D space), geometric elements may be points, lines, planes, cylinders, and spheres. This problem is central to many applications, such as computer-aided design (CAD) [2], molecular modeling, localization in wireless sensor networks [3], dynamic geometry [4] and many others. Solving a GCS consists of placing the geometric elements in the Euclidean space, in such a way that the constraints are satisfied. Assigning coordinates to the geometric elements of

a GCS that satisfies the constraints was called by Hendrickson a realization problem [5], Saxe [6] has shown it to be NP-hard. Many solvers have been proposed in the literature: graph-based, symbolic, numerical, and rule-oriented. For furthers details, please see the survey of Bettig and Hoffmann[7] and Jermann et al.[8]. If a GCS is incomplete, i.e., there are not enough constraints between geometric elements, then it will be called under-constrained. If the specified constraints are conflicting, i.e., there are too many constraints defined between geometric elements, it will be called over-constrained. A GCS is called well-constrained, if it has a finite number of solutions. We formally define those notions in chapter 2.

In this Thesis, we restrict ourselves to 2D geometric constraint systems in the generic sense; we focus only on the solvability of the constraint graph and ignore the numerical values of the geometric constraint. Particularly, we focus on graph-based methods, developed in Computer-Aided Design context, such as those presented by Ait-Aoudia and Foufou[9], Ait-Aoudia et al.[10], Fudos and Hoffmann[11], Hoffman et al.[12], Latham and Middleditch[13], Owen[14]. Many such solvers transform the GCS into a graph. By applying some decomposition techniques on the constraint graph, they isolate under, over, and well-constrained parts. The well-constrained part is then analyzed by a decomposition method to find the small, solvable subgraphs. The final solution is produced by merging the solved subgraphs in respect to an order of resolution generated in the decomposition phase. This is referred to as Decomposition/Recomposition plan (DR-Plan) (Hoffman et al. [15]). The primary aim of this decomposition is to speed up the resolution process by limiting the use of the direct algebraic methods to subsystems that are as small as possible. In [15], Hoffman et al. classify DR-planners into two main categories: SR-Planners (constraint shape recognition), and MM-planners (generalized maximum matching). Examples of SR-Planners are given in Joan-Arinyo et al. [16], Fudos and Hoffmann [11] and Owen [14]. These solvers

are not complete, i.e., they do not solve every well-constrained graphs, but only a subclass of geometric configurations. These methods require that the constraint graph is biconnected [16], and the smallest solvable subgraphs are triangles [15]. Those algorithms can be theoretically extended to handle other types of shapes than triangles by giving a particular recognition algorithm to each new kind of shape. We note that there is an infinite collection of shapes as reported by Joan-Arinyo et al. in [17]. As a result, SR-planners cannot be complete. The second class is referred to as MM-planner (see for example Ait-Aoudia et al. [10], Hoffman et al.[12], and Ait-Aoudia and Foufou[9]). In such type of planner, there is no restriction on the domain of geometric constraint configurations, and the smallest subgraph can be any non-reducible well-constrained graph. These categories use a Maximum Matching or a flow based algorithm to detect small, solvable subgraphs and their order of resolution. Those methods are complete, i.e., any well-constrained graph can be decomposed, solved and recomposed.

The decomposition of the 2D geometric constraint graph is the goal of many geometric constraints solver, it speeds up the resolution process and makes CAD software more interactive. The majority of geometric constraint solver (also called planners) proposed in the literature discuses some specific properties of the constraints graphs: the size of the largest subgraph, the number of subgraphs, the class of the graph (Solvable by SR or MM-planners). However, there is no known datasets or tools that can be used for analyzing the performances of graph-based solvers. This is due to many difficulties, notably, until now, no known algorithm or method can generate graphs of a particular class, with the desirable properties presented above. Another problem is the construction of the non-decomposable well-constrained graph. This category of graphs is tough to be manually constructed. Building a non-decomposable graph of ten nodes is a challenging task. The subject of our research is to provide some answers to those questions.

(a) (b)

FIGURE 1.1: Two well-constrained graphs: (a) solvable by SR-Planners and (b) solvable only by MM-Planners

1.2 Objectives and contributions

Most research done in 2D geometric constraint solving is limited to a theoretical discussion and analysis. Algorithms are always tested and compared using some well-chosen examples. This limitation is due to the absence of any standard dataset that represents geometric constraints problems with different properties. As far as we know, there is no known method for the automatic generation of two-dimensional GCS instances with representative properties. This is due to many difficulties that will be answered in this thesis. Our research goal is the design of a generator of 2D geometric constraint graphs, a tool that can be used for the creation of desirable synthetic scenarios, i.e., generation of many situations of difficulties that can be used for analyzing the performances of geometric constraint solvers. This tool can generate 2D geometric constraint graph with desired structural properties, such as the size of the largest subgraph, the number of subgraphs and the class of the graph (Solvable by SR or MM-planners). We focus on generating two-dimensional well-constrained geometric constraint graphs, where edges represent distance constraint, and for which the following Laman condition [18] holds:

In 2D space, a geometric constraint graph $G = (V, E)$ where $|V| = n(n > 1)$ and $E = m$ is structurally well-constrained if and only if $m = 2n - 3$ and $m' \leq 2n' - 3$ for any induced subgraph $G' = (V', E')$, where $|V'| = n'(n' > 1)$ and $|E'| = m'$.

This condition gives the necessary condition of generic solvability for any 2D GCS. This is very significant for making consistent tests or observing algorithm behavior on graphs with various sizes and structural properties, covering different situations of difficulties and sizes. The design of this tool is based on the classification of different solvers proposed in the literature, such as those presented in [15]. The proposed generator verify the following properties:

1. **Completeness:** it can generate any possible geometric configuration. i.e., the generator covers all the domain of 2D geometric constraint problems.

2. **Customizable:** it can generate some specific configurations, this is done by parameterizing the generator for producing graphs that are solvable by either SR-Planners or only by MM-Planners. Moreover, it can generate a specific domain of a 2D GCS or a general one. By specifying what we called a degree of the decomposability, we can build a graph that has a low or a high number of solvable subgraphs. We can also specify the average size of the smallest subgraph. Those two parameters represent our adopted metric of solvability.

3. **Simplicity and efficiency:** our generator is easily understood; it can be straightforward implemented an it is based on a verified theory.

As far as we know, no such tool was proposed before. Henneberg [19] proposed a construction method for use by engineers and architects in the building of large statically rigid frameworks from smaller ones. We have introduced the relation between the Henneberg construction and the decomposability of the 2D geometric constraint graph. To the best of our knowledge, for the first time the following contributions have been successfully done:

1. There is no known algorithm for the creation of large non-decomposable well-constrained graphs. This type of graphs is tough to be manually constructed: building a non-decomposable graph of ten nodes is a challenging task. This category of graphs was well discussed in the constraint CAD literature. Our proposed generator is the first tool that can easily generate this type of graphs.

2. With the proposed generator, it is very easy to create a 2D geometric constraint graphs with respect to the classification and properties proposed by the research communities in this area, especially, graphs that belong to SR-planner or MM-planner domain (see Hoffman et al. [12] for more detail). i.e., with our generator, it is very easy to create graphs that belong to a particular domain of GCS.

3. Performance analysis of 2D geometric constraints solver can be done in an easy way with our generator. Two metrics can parametrize this tool: the size of the largest subgraph as proposed in [15] by Hoffman et al. and a new metric that we have introduced: the degree of decomposability. Those two metrics represent the essential structural properties of constraint graphs that are in relation with the design of many solvers proposed in the literature.

1.3 Thesis organization

The remaining of this thesis is organized as follows:

Chapter 2 presents some background on geometric constraints solving and their importances. We briefly recall some common concepts from graph theory; then we discuss the rigidity theory. We terminate this chapter by giving some others interesting applications of the geometric constraints solving.

Chapter 3 presents the different approaches of 2D geometric constraint solving, that arise from CAD Application: the numerical and the logic-based methods.

In chapter 4, we show some interesting algorithms that use graph-based methods, the constructive solvers, and the general solvers.

Our contribution, the proposed generator of 2D well-constrained graphs and some tests on their decomposability is presented in Chapter 5.

Chapter 6 summarizes the study and draws conclusions regarding the usefulness of the proposed 2D geometric constraint generator.

Chapter 2

Background

Contents

2.1 Geometric constraints solving: problem definitions 9

2.2 Modelling a GCS by a system of equations 11

2.3 Graph theory 13

2.4 The importance of GCS decomposition in CAD . 16

2.5 NP-Hardness of finding real solution in 2D . . . 19

2.6 Rigidity theory and Laman theorem 20

2.6.1 Laman condition 21

2.6.2 Rigidity testing in 2D: The pebble game 23

2.7 Generalization of Laman's theorem to other types of constraints 27

2.8 Combinatorial characterization of the 3D case . . 29

2.9 Other application of geometric constraints solving 31

2.9.1 Localisation in Wireless Sensor Networks 31

2.9.2 Molecular Biology 32

2.9.3 Dynamic Geometry 33

This chapter provides the necessary background for the thesis. We start by some important definitions, and we recall some concepts from graph theory. We highlight the importance of geometric constraint solving in CAD. We study the rigidity theory, which is in the center of 2D graph based solvers. We terminate with some important applications of geometric constraints solving.

2.1 Geometric constraints solving: problem definitions

Definition 2.1. A *constraint* is the desired relationship between several unknowns (or variables), each takes a value in a given domain. Thus, a constraint restricts the values that variables can take.

Definition 2.2. A *geometric constraint* is a relationship between two geometric objects or entities. We can cite, for example, the distance between two points, the angle between two lines, coincidence between a point and a line, or the tangency between circles, etc.

Definition 2.3. The *degree of freedom* (DOF) of a geometric entity is the number of independent coordinates used to represent it. It is equal to two for points and lines in $2D$. The number of DOFs of a geometric constraint is the number of independent equations needed to represent it. In $2D$ geometry, a constraint cancels one degree of freedom, and the geometric elements have different degrees of freedom. A point has two DOFs that corresponds to the translation along the two directions. A line also has two DOFs, one for translation and another for rotation. A circle with fixed radius has two translational DOFs. Any rigid body in the plane, like a triangle, except a single point or a single line, has three DOFs in the plane.

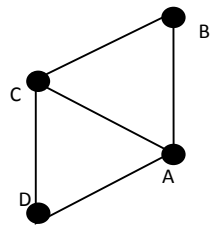


FIGURE 2.1: A well-constrained GCS in 2D.

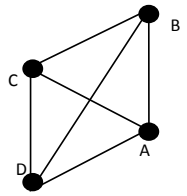


FIGURE 2.2: An over-constrained GCS in 2D.

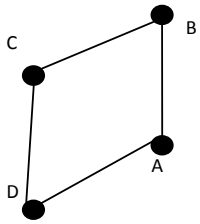


FIGURE 2.3: An under-constrained GCS in 2D.

Definition 2.4. A geometric constraint system (GCS) is given by a set of geometric elements, along with required relationships, called the constraints.

Definition 2.5. A geometric constraint system S is well-constrained if it has a finite number of solutions. Figure 2.1 shows an example of a well-constrained problem in 2D space.

Definition 2.6. A geometric constraint system S is over-constrained if it has no solution. Figure 2.2 shows an example of an over-constrained problem in 2D space.

Definition 2.7. A constraint system S is under-constrained if it has an infinite number of solutions. See figure 2.3 for an example of an under-constrained problem in 2D space.

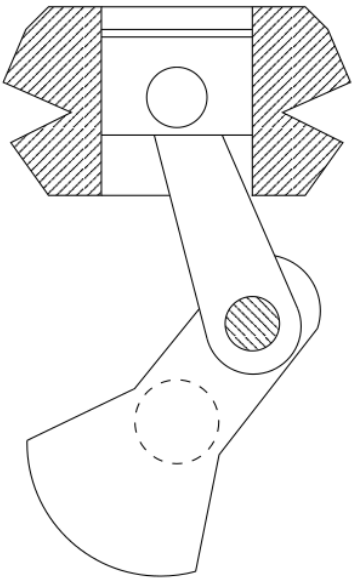


FIGURE 2.4: A piston-crankshaft mechanism.

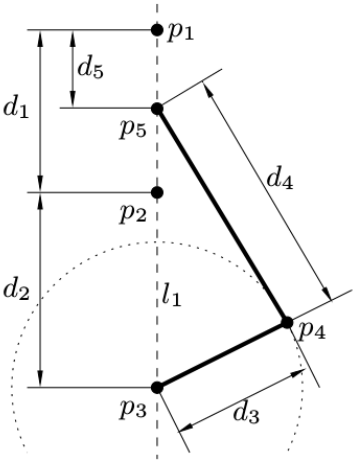


FIGURE 2.5: Geometric constraint system of the piston-crankshaft mechanism

2.2 Modelling a GCS by a system of equations

A GCS can be translated into a system of non-linear algebraic equations, which can be solved using iterative methods. Figure 2.5 presents a piston-crankshaft mechanism taken from [2], its corresponding 2D GCS is given in figure 2.4. This problem comprises five points $p_i, 1 \leq i \leq 5$, and a line l_1. Its corresponding system of equations is well-constrained. The set of geometric

elements consists only of lines and points. Table 2.1 gives the set of geometric constraints and their values.

A distance constraint d between two points $p_i(x_{p_i}, y_{p_i})$ and $p_j(x_{p_j}, y_{p_j})$ in 2D is written as:

$$(x_{p_j} - x_{p_i})^2 + (y_{p_j} - y_{p_i})^2 = d_{ij}^2$$

For a line l_1, defined by the triplet : (r_1, n_{1_x}, n_{1_y}), where r_1 is Euclidean signed distance of the origin from the line, (n_{1_x}, n_{1_y}) is a normal vector of the line, normalized, so it holds: $n_{1_x}^2 + n_{1_y}^2 = 1$

If d is a signed distance between a line l_1 and a point $p_1(x_1, y_1)$, then the corresponding equation is defined by:

$$(x_1, y_1).(n_{1_x}, n_{1_y}) = r_1 + d, \text{ i.e., } x_1 n_{1_x} + y_1 n_{1_y} = r_1 + d$$

We can write the algebraic equations of the piston-crankshaft problem in the manner shown in the formula 2.1. This system of equations can be solved using any numerical method similar to those presented next in section 3.1.

Pair of geometric elements	Type of constraints	Value of the constraint
(p_1, p_2)	dpp	d_1
(p_2, p_3)	dpp	d_2
(p_3, p_4)	dpp	d_3
(p_4, p_5)	dpp	d_4
(p_5, p_1)	dpp	d_5
(l_1, p_1)	dpl	0
(l_1, p_2)	dpl	0
(l_1, p_3)	dpl	0
(l_1, p_5)	dpl	0

TABLE 2.1: A set of geometric constraints for the piston-crankshaft mechanism.

$$\begin{cases} (x_{p_2} - x_{p_1})^2 + (y_{p_2} - y_{p_1})^2 = d_1^2 \\ (x_{p_3} - x_{p_2})^2 + (y_{p_3} - y_{p_2})^2 = d_2^2 \\ (x_{p_4} - x_{p_3})^2 + (y_{p_4} - y_{p_3})^2 = d_3^2 \\ (x_{p_5} - x_{p_4})^2 + (y_{p_5} - y_{p_4})^2 = d_4^2 \\ (x_{p_5} - x_{p_1})^2 + (y_{p_5} - y_{p_1})^2 = d_5^2 \\ x_{p_1} n_{1_x} + y_{p_1} n_{1_y} = r_1 \\ x_{p_2} n_{1_x} + y_{p_2} n_{1_y} = r_1 \\ x_{p_3} n_{1_x} + y_{p_3} n_{1_y} = r_1 \\ x_{p_4} n_{1_x} + y_{p_4} n_{1_y} = r_1 \end{cases} \tag{2.1}$$

2.3 Graph theory

Graph theory provides a powerful concept for the structural analysis of geometric constraint problems. Many of graph theory concepts are defined differently by different authors. In this section, we briefly recall some standard concepts from graph theory, for more information on treatment of these concepts, we refer the reader to Thulasiraman and Swamy [20], from where the following definitions are adapted.

Graph: A graph $G = (V, E)$ consists of two sets: a finite set V of elements called vertices and a finite set E of elements called edges. Each edge is identified with a pair of vertices. If the edges of a graph G are identified with ordered pairs of vertices, then G is called a directed or an oriented graph. Otherwise, G is called an undirected. We are concerned with undirected graphs. We use the symbols $v_1, v_2, v_3, ...$to represent the vertices and the symbols $e1, e2, e3, ...$ to represent the edges of a graph.

Incidence, adjacency and degree: An edge is said to be incident on its end vertices. Two vertices are adjacent if they are the end vertices of some edge. If two edges have a common end vertex, then these edges are said to be adjacent. The degree (valency) of a node n_i of a graph, denoted by $deg(n_i)$, is the number of members incident with that node.

Cardinality and size: The number of vertices, the cardinality of V, is called the size of the graph and denoted by $|V|$. We usually use n to denote the size of G. The number of edges, the cardinality of E, is denoted by $|E|$. We usually use m to denote it.

Subgraph: Consider a graph $G = (V, E)$. $G' = (V', E')$ is a subgraph of G if V' and E' are, respectively, subsets of V and E such that an edge (v_i, v_j) is in E' only if v_i and v_j are in V'.

Path and connectivity: A path is a sequence of vertices $v_1, v_2, ..., v_n$ such that (v_i, v_{i+1}) is an edge for $1 \leq i \leq n$. A path is simple if all vertices on the path are distinct. A graph $G = (V, E)$ is connected if there exists a path between every pair of vertices in G, otherwise, G is disconnected. The maximal connected subgraphs of a disconnected graph G are the connected components of G.

Walk, trail and circuit: A walk in a graph $G = (V, E)$ is a finite sequence of vertices $v_0, v_1, v_2, ..., v_k$, such that (v_{i-1}, v_i) is an edge in the graph G. A walk is open if its end vertices are distinct; otherwise, it is closed. A walk is a trail if all its edges are distinct. A trail is open if its end vertices are distinct; otherwise, it is closed. An open trail is a path if all its vertices are distinct. A closed trail is a circuit if all its vertices except the end vertices are distinct.

Acyclic graph and spanning trees: A graph is said to be acyclic if it has no circuit. A tree is a connected acyclic graph. A tree of a graph G is a connected acyclic subgraph of G. A spanning tree of a graph G is a tree of G having all the vertices of G. A connected subgraph of a tree T is called a subtree of T. The edges of a spanning tree T are called the *branches* of T, and those which are not in T are called *chords*.

Planar graph: A graph is planar if it can be drawn in a plane without graph edges crossing.

Bipartite graph: A graph $G = (V, E)$ is called bipartite if V can be partitioned into two sets X and Y such that each edge in E has one endpoint in X and one endpoint in Y. X and Y are then called the vertex classes of G. If the partitioning of the vertices into vertex classes is explicit, we use the notation $G = (X, Y, E)$.

Graph matching: Given a graph $G = (V, E)$, a matching M in G is a set of pairwise non-adjacent edges; that is, no two edges share a common vertex. A vertex is matched (or saturated) if it is an endpoint of one of the edges in the matching. Otherwise, the vertex is unmatched. A matching M of a graph G is maximal if every edge in G has a non-empty intersection with at least one edge in M.

Maximum and perfect matching: A maximum matching is a matching that contains the largest possible number of edges. A perfect matching is a matching which matches all vertices of the graph. That is, every vertex of the graph is incident to exactly one edge of the matching.

Alternating and augmenting path: An alternating path is a path in which the edges belong alternatively to the matching and not to the matching. An augmenting path is an alternating path that starts from and ends on free (unmatched) vertices.

Dulmage–Mendelsohn decomposition: In graph theory, the Dulmage-Mendelsohn decomposition is a partition of the vertices of a bipartite graph into subsets, with the property that two adjacent vertices belong to the same subset if and only if they are paired with each other in a perfect matching of the graph. It is named after A. L. Dulmage and Nathan Mendelsohn, who published it in 1958 [21].

2.4 The importance of GCS decomposition in CAD

Generally, in most CAD software, a $2D$ geometry is considered the starting-point for most $3D$ models. First, the designer sketches a $2D$ draft, then adds some relation between geometric elements. The geometric constraint solver transforms the problem into a geometric constraint graph (GCG). After the solving process, that can be done in real-time, geometric objects are adjusted to conform to the specified constraint. This offers the advantage of freeing the user from the tedious task of the exact location of the different geometric elements. The $2D$ sketch is extruded to obtain the final $3D$ model. Geometric models will be easily updated in the future, by simply modifying the values of the different constraints. Because the underlying system of equations is non-linear, and most solving methods are $O(n^3)$ or worse, the resolution speed depends on the size of the largest system of equations. Decomposing the GCS, which is the central role of the planner, will speed up the resolution

process and make the CAD software more interactive. Thus, the productivity of the designer will be increased.

To illustrate the importance of planners, we use the example of figure 2.6 taken from [22]. The problem consists of 13 circles, eight lines, which are connected by 19 tangency constraints: circle-circle and circle-line, four constraints of angles between lines and 16 distance constraints. Each geometric element has two unknown: x_i and y_i. Circles are defined by the two coordinates of their center; a line can also be defined by two values: x_i, the slope of the line and y_i its distance from the origin. The corresponding constraint graph is given in figure 2.7, nodes are numbered from 1 to 21, and each constraint is represented by an edge of the graph. We have decomposed its corresponding graph using

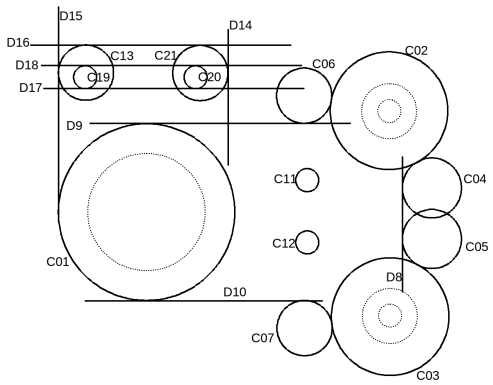


FIGURE 2.6: A 2D constrained model.

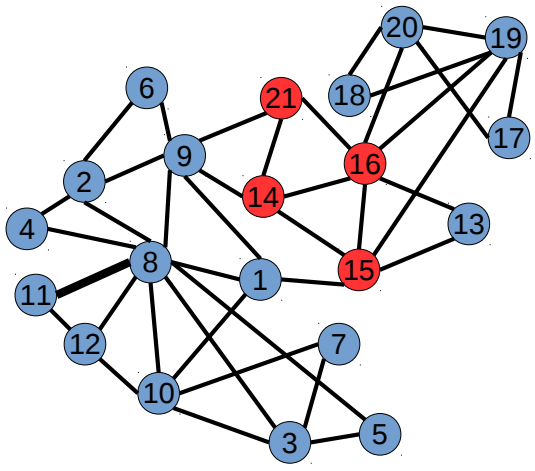


FIGURE 2.7: The corresponding constraints graph.

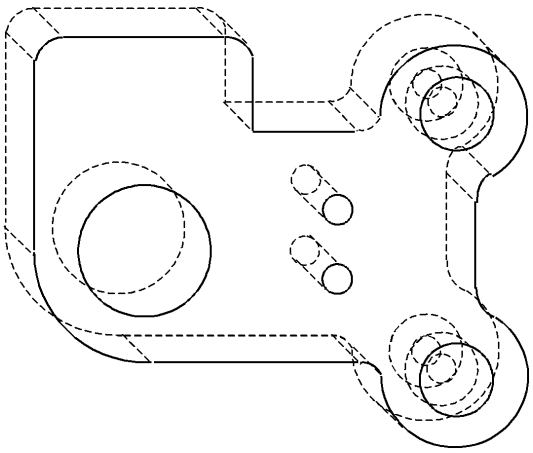


FIGURE 2.8: The extruded model.

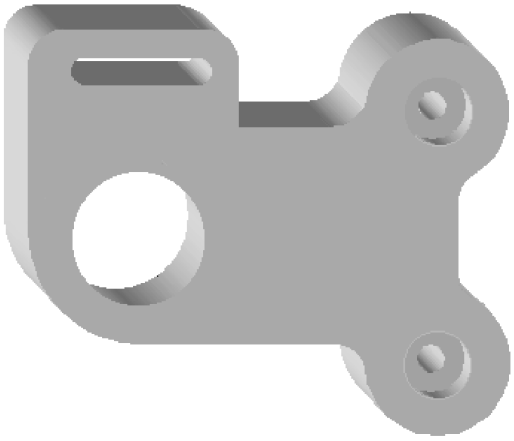


FIGURE 2.9: The final 3D model.

the method presented in [10]. The result gives 20 systems of equations, the largest of which contains eight unknowns. To be solved, three unknown must be fixed: x_{11}, y_{11} and x_8. Geometrically, this correspond to putting the center of the circle $C11$ in the origin $(0, 0)$ and fixing the slope of the line $D8$ to 1, i.e., its normal vector will be $(0, 1)$. The set of unknowns for each equation and the order in which they will be solved is given by the following ordered set:

$$\{\{x_{11}\}, \{y_{11}\}, \{x_8\}, \{y_8\}, \{y_{12}, x_{12}\}, \{x_{10}, y_{10}\}, \{x_1, y_1\}, \{x_3, y_3\}, \{x_5, y_5\}, \{y_7, x_7\},$$
$$\{x_9, y_9\}, \{x_2, y_2\}, \{y_4, x_4\}, \{x_6, y_6\}, \boldsymbol{\{x_{14}, y_{14}, x_{21}, y_{21}, y_{16}, x_{16}, y_{15}, x_{15}\}}, \{y_{13}, x_{13}\},$$
$$\{x_{19}, y_{19}\}, \{y_{20}, x_{20}\}, \{y_{17}, x_{17}\}, \{y_{18}, x_{18}\}, \}$$

Without the decomposition process, we have to solve a quadratic system of 42 unknowns. The time required for finding a solution will significantly decrease by decomposing the system into smaller subsystems, and consequently, the design process becomes more interactive.

Figure 2.8 and 2.9 show how the 3D model is constructed by extrusion. In chapter 4, we present some interesting results on geometric constraint graph decomposition.

2.5 NP-Hardness of finding real solution in 2D

Assigning coordinates to the geometric elements of a GCS that satisfies the constraints is called by Hendrickson a realization problem [5]. Saxe [6] has shown it to be NP-hard, he says in the conclusion of his paper: *"you are trying to solve the wrong problem." Rather than looking for an efficient worst-case algorithm, it would be more promising to seek an algorithm that gives good performance in cases which arise in practice.* Fudos and Hoffmann [11] have reduced one of the first problems that was proven to be NP-complete, the SAT problem, to the geometric real solution problem. They also proved the NP-Hardness for a sub-class of ruler and compass constructible GCS. All the following sub-problems derived by imposing any combination of these restrictions are also proved to be NP-hard [11]: the set of geometric objects consists only of lines and points; the set of geometric constraints consists only of distances and angles; the domain of the value of the geometric constraints is {0, 1, 2}; and the geometric problem is well-constrained.

2.6 Rigidity theory and Laman theorem

In this section, we draw on the powerful results from rigidity theory and combinatorial theory. We show that the geometric constraint problem is well understood in two dimensions; however, only partial similar results are available in three dimensions.

A set of rigid bars connected by hinges allowing full rotation of the incident bars is called a framework (or bar-joint framework), rigidity theory is a branch of mathematics that studies whether a framework is rigid or not (see [23] for more details). A framework is not rigid if you can move hinges continuously, such that bar lengths are preserved, which leads to a deformation of the framework. In this section, we show the relevant results in rigidity theory, the combinatorial rigidity of graphs and the pebble game [5], an algorithm for testing rigidity of frameworks. We note that most of the known results on the rigidity of frameworks are valid only in the two-dimensional case.

Definition 2.8. A framework (G, p) is the combination of a graph $G = (V, E)$ and a map $p : V \to \mathbb{R}^d$. (G, p) is rigid if all continuous deformations compatible with constraints are rigid body motions, i.e. a composition of translations and/or rotations.

The rectangle is an example of a flexible bar and joint framework (see 2.10a). By moving the top two vertices, the rectangle will be deformed to a parallelogram, the length of diagonals is changed. Adding an extra bar (see figure 2.10b) makes this frame rigid, the distances between all pairs of seals remain fixed. The addition of another bar in a rigid part (see 2.10c) is unnecessary and the frame becomes over-rigid (also called stressed).

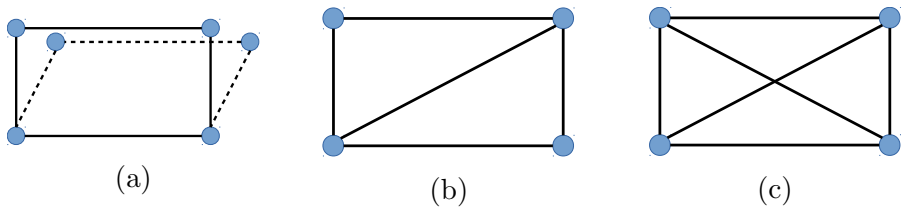


FIGURE 2.10: Flexible, rigid and stressed framework.

For our purpose, a framework provides a natural high-level model for geometric constraint solving in real two-dimensional space.

2.6.1 Laman condition

The following theorem, due to Laman [18], gives the necessary combinatorial condition for generic minimal rigidity in 2D space. Laman's theorem was the first combinatorial characterization of generic rigid graphs in two dimensions space.

Theorem 2.9. *[18] A graph $G = (V, E)$ is minimally rigid in $\mathbb{R}^2$ if and only if $|E| = 2|V| - 3$ and for every subgraph $G'(E', V')$, $|E'| \leq 2|V'| - 3$.*

For the condition $|E| = 2|V| - 3$, 2 represents the DOFs of a point in 2D space, and 3 represents the DOFs of a rigid body, two translational and one rotational. We know that a vertex in 2D has two DOFs, a graph is rigid means that all DOFs are removed by the constraints. Each constraint is represented by an edge, i.e., an edge remove a single DOF. The graph has $|V|$ vertices, which means we have $2|V|$ DOFs. The graph has $2|V|-3$ edges, which implies that $2|V|-3$ DOFs are removed. The remaining 3 DOFs represent the DOFs of the whole graph. Informally, to be rigid a graph needs at least $2n - 3$ *independent* edges. If a subgraph $G'(E', V')$ has more edges than necessary, i.e., $|E'|>2|V'| - 3$, then some edges are dependant. Non-redundant edges are independent, i.e., they remove a degree of freedom each. Hence, $2n - 3$

independent edges guarantee rigidity. Figure 2.11 gives three minimally rigid graphs, all of them satisfy the Laman condition. Figure 2.12 gives four graphs that are not minimally rigid, for example figure 2.12d has $|E| = 2|V| - 3$ but is not minimally rigid because it contains a subgraph $G'(V', E')$ where $|E'|>2|V'| - 3$.

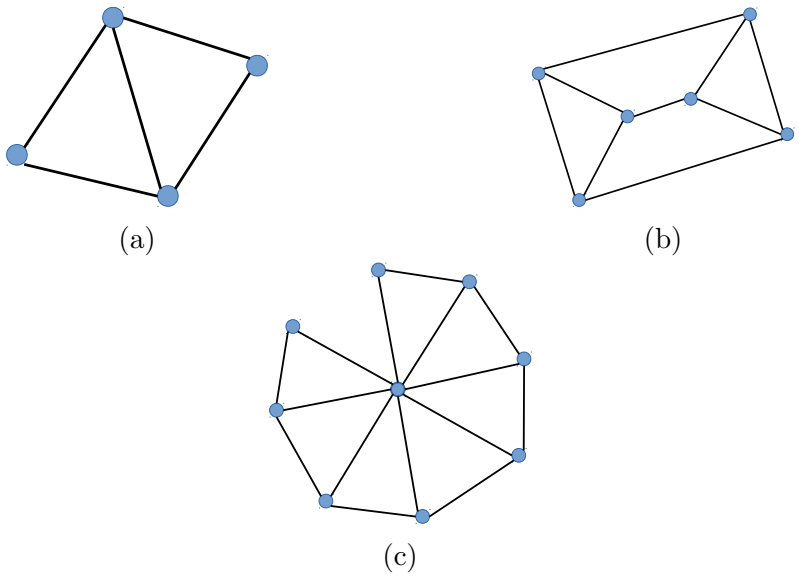


FIGURE 2.11: Examples of minimally rigid Graphs

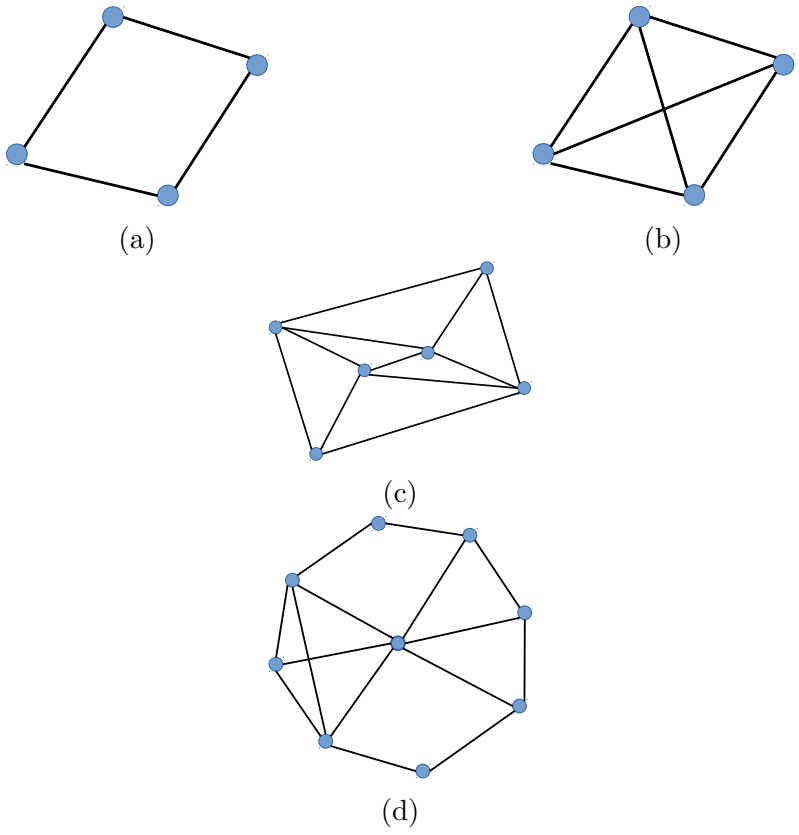


FIGURE 2.12: Examples of non-minimally rigid graphs. (a) is not rigid. (b), (c) and (d) are over-rigid, i.e., some edges are dependant

2.6.2 Rigidity testing in 2D: The pebble game

The Laman condition is a purely counting condition. Its direct use for testing generic rigidity of graphs gives an exponential algorithm. It requires counting the edges in every subgraph, which are in exponential number. Many polynomial algorithms have been proposed for testing the rigidity of graphs. Imai [24] and Hendrickson [5] presented an $O(n^2)$ algorithm using a network flow approach. Jacobs and Hendrickson [25] have simplified Hendrickson's algorithm to make it more usable. A brief overview of the pebble game algorithm [25] will be given here. Let us start by the following theorem that represents an alternative version of Laman's theorem.

Theorem 2.10. *[25] For a graph $G(V, E)$, the following are equivalent.*

1. *The edges of G are independent in two dimensions.*

2. *For each edge (v_i, v_j) in G, the graph formed by adding three additional edges (v_i, v_j) has no induced subgraph G' in which $|E'|>2|V'|$.*

In 2D space, to test the rigidity of a graph $G(V, E)$, we have to find $2|V| - 3$ independent edges. We start by a graph $G'(V, U)$ where U is initially empty. The idea behind Jacobs and Hendrickson algorithm is to grow a maximal set of independent edges, one at a time. Each candidate edge $e = (v_i, v_j) \in E$ is quadrupled, and the resulting graph is tested using the alternative version of Laman's theorem. If the edge e is independent, then it will be added to the graph U. This leads to an incremental construction of the independent edges set U. This process is done using a pebble game [25], that starts by associating for each vertex $v \in G$, two pebbles, each pebble represents a DOF. Hence, we have $2n$ pebbles in total. We say that an edge is covered if it has a pebble placed on either of its ends, we recall that an edge eliminates exactly one DOF. The goal of the pebble game is to cover all the edges of the graph, i.e., test if the edges of the graph G are well distributed in such a way that they remove every DOF of every vertex. To test if a new edge is not redundant in the existing set, we have to quadruple the edge and find a pebble covering for the four new edges. Figure 2.13 shows a cover testing for a simple graph of three edges. Figure 2.14 shows the breath first search for a free pebble, each vertex in the graph of figure 2.14a is labelled by the minimum search length from the starting vertex. (step 9 of Algorithm 1). Figure 2.14b shows the pebbles rearrangement (or swapping) after a free edge is found (step 11 of Algorithm 1). This algorithm has an $O(|V||E|)$ time complexity. We note that if the pebble game detects that the graph G is not rigid, it can discover rigid subgraphs, and if it is rigid but not minimally-rigid, redundant edges will be returned.

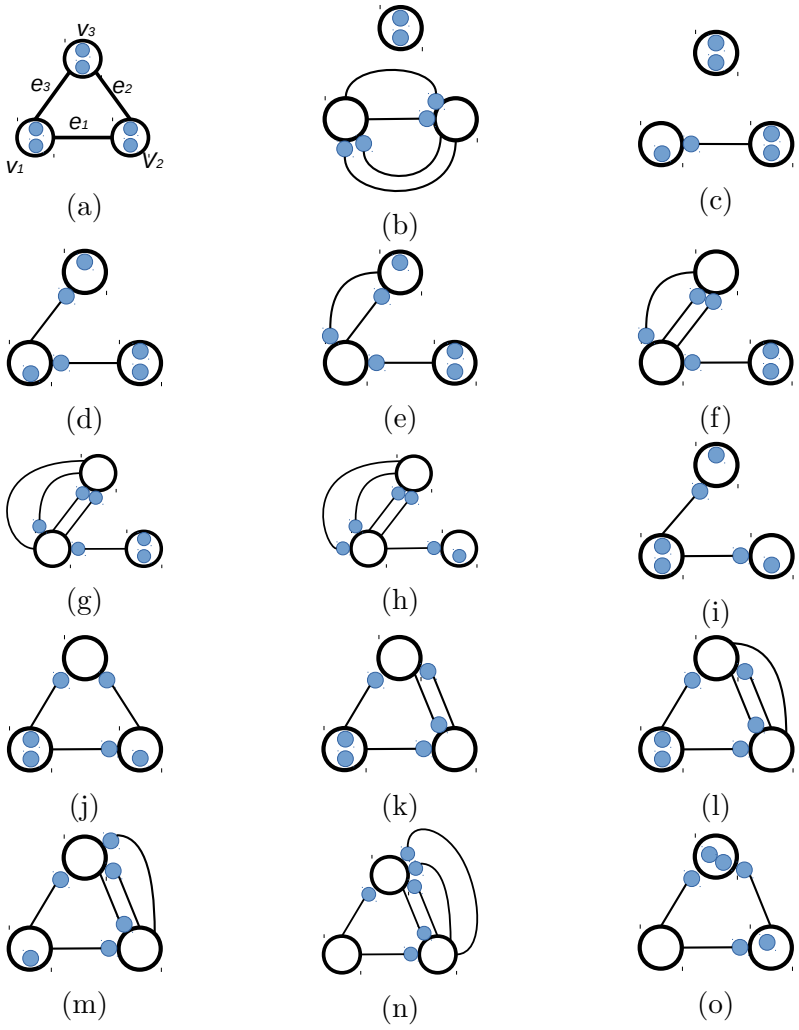


FIGURE 2.13: Illustration of the pebble game with $|V| = 3$

Covering progression from left to right, from top to down. (a) the initial assignment, 2 pebbles for each vertex. (b) the edge e_1 is quadrupled and tested for independence.(c) edge e_1 is covered. (d) the first copy of e_3 is covered. (e) the second copy of e_3 is covered.(f) the third copy of e_3 is covered. (g) the forth copy of e_3 is added but there is no pebble in its adjacent vertices : v_1 and v_2, a pebble search will be started using a depth first search from v_3 . (h) a free pebble is found in v_2 it will be shifted to v_1. (i) the edge e_3 is covered, four pebble remain. (j),(k),(l),(m),(n),(o) The same process will be repeated for the edge e_2, when covered, three pebble remain(two in v_3 and one in v_2), they represent the DOFs of the whole graph.

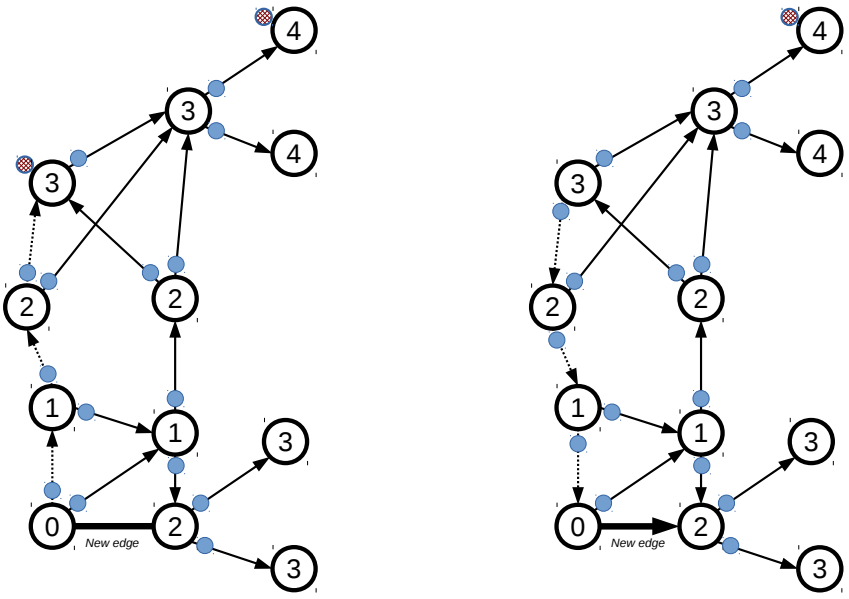
Algorithm 1 Jacobs and Hendrickson algorithm: The pebble game.

Input: $G(V, E)$: a 2D geometric constraint graph with $|E| = 2|V| - 3$;

Output: $(rigid, U)$ ▷ $rigid$: a boolean, $true$ if G is rigid, $false$ if not rigid; U: the set of independent edges.

```
procedure TestRigid(G)

    for each v ∈ V do
        Assign 2 free pebbles to v.
    end for
    U ← ∅
    for each edge (v_i, v_j) ∈ E do
        Quadruple (v_i, v_j)
        Search for 4 free pebbles to cover quadrupled edges (v_i, v_j).
        if found then
            Swap pebbles until the 4 edges are covered.
            U ← U ∪ {(v_i, v_j)}
            if |U| = 2|V| − 3 then
                return (true, U)
            end if
        end if
    end for
    if |U|<2|V| − 3 then
        return (false, U)
    end if
end procedure
```



(a) A breath first search for a free pebble. (b) Pebble rearrangement

FIGURE 2.14: Searching for free pebble

2.7 Generalization of Laman's theorem to other types of constraints

Major graph-based solvers are based on Laman's theorem, which is valid only for point-point distance constraint. The introduction of new types of constraints like the angle between lines, tangent between circles, alignment of points, or variable-radius circles, which are widely present in CAD software, always need some verifications and pre-processing, for each particular case, before the application of the graph-based decomposition techniques. The generalization of graph-based approaches to other geometric elements than point-point distance can give good results when applied to GCS of a small or medium size, but when the GCS grow up, it can lead to erroneous results [26]. This problem is due to the geometric theorems. This mean that some hidden relations between geometric elements will be present without an explicit constraint between them. Michelucci and Foufou proposed in [26, 27] a method called the witness configuration method that detects all kinds of dependences:

structural dependences already detectable by graph-based methods, but also non-structural dependences, due to known or unknown geometric theorems, which are undetectable by graph-based methods. To illustrate this problem, the following 2D examples are given.

- **Line-line angles:** If we have for example three lines l_1 , l_2 , and l_3 and the angles $(l_1, l_2) = \alpha$ and $(l2, l3) = \beta$, then the angle between (l_1, l_3) must be $\gamma = \pi - \alpha - \beta$ (see figure 2.15). If the user specifies only two angle constraints, we can think to add the third as "implicit" constraint. This is not true; the GCS is not rigid because an homothetic transformation is always possible. This problem can also happen when n lines are constrained with $n - 1$ angle constraint. In conclusion, angle cycles are only sources of hidden dependences not detectable by graph-based solvers. A geometric angle constraint system, structurally well-constrained graph can be over-constrained in a geometric sense. There is no combinatorial characterization of line-line angle constraint that immediately lead to efficient algorithms[28].

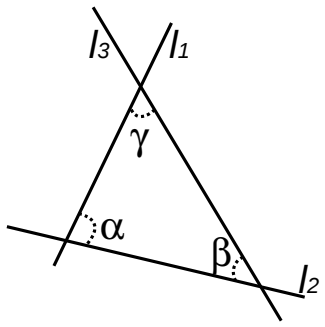


FIGURE 2.15: Three lines with angle constraint

- **Variable-radius circle:** Fixed-radius circles can be replaced equivalently by points. Tangency between circles of the known radius can be directly transformed to a point-point distance. The circle placement problem is reduced to positioning the center point. Hence, this case can be easily incorporated into graph-base solvers [11]. Variable-radius

circle is a circle with three unknowns (or DOFs): the two coordinates of its center $c(x, y)$ and the radius r. Those three unknowns must be deduced from other constraints. It is a more complicated case, examples of works dealing only with this issue are [29, 30]. Solving this problem by graph-based approaches requires a special handling of several particular cases, each of them relates to the way in which the circle is connected to other geometric elements. With numerical solvers, variable-radius circle pose no particular problem [29].

- **Point-line incidence:** in 2D space, a combinatorial characterization for well-constrained systems of point-line incidence constraints is terribly difficult. An example of dependencies, taken from [26], is Pappus theorem: if p_1, p_2, p_3 are three distinct aligned points, and if q_1, q_2, q_3 are three distinct aligned points, then the three intersection points $p_1q_2 \cap p_2q_1, p_1q_3 \cap p_3q_1$, and $p_2q_3 \cap p_3q_2$ are aligned as well [26](see figure 2.16).

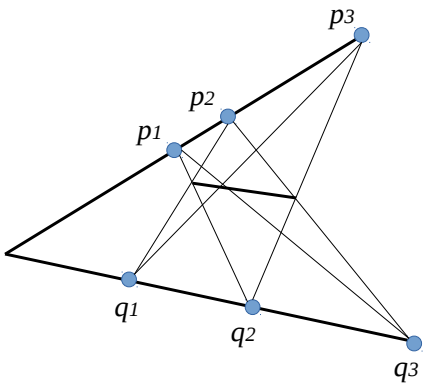


FIGURE 2.16: Pappus Theorem

2.8 Combinatorial characterization of the 3D case

Laman's [18] condition is necessary and sufficient in 2D space. A false intuitive extension of the Laman condition to 3D space can be as follows: the graph

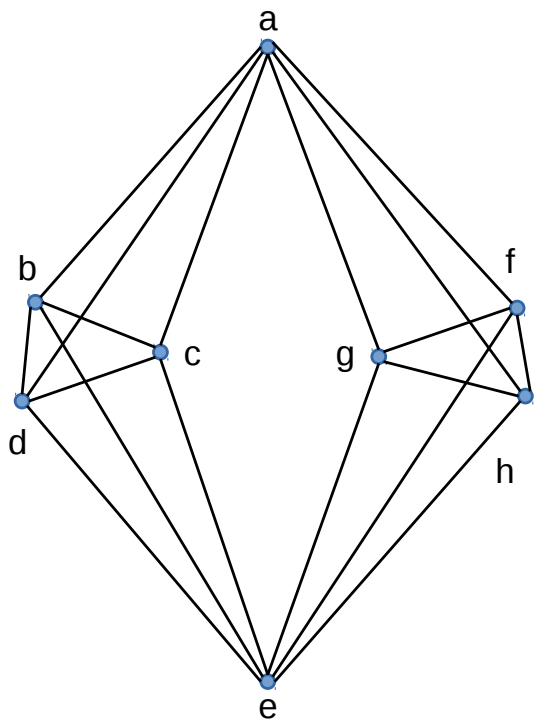


FIGURE 2.17: The double banana, a 3D counter example.

$G(V, E)$ is minimally rigid if it has $3|V| - 6$ edges and for every subgraph $G'(E', V')$, $|E'| \leq 3|V'| - 6$. 3 represents the three DOF of the point in 3D space, 6 represents four translational and two rotational DOFs of the whole graph in 3D space. This condition is necessary but not sufficient for rigidity in 3D. The Laman condition in 2D can not be extended to 3D. Figure 2.17, called the double banana , is a classic well-known counter-example. The two banana units are free to rotate around the axis through the two point a and e, i.e., this graph is not rigid in 3D space but satisfies the Laman condition.

The generalization of Laman's theorem to dimensions three, or higher, has been proved to be incorrect. Until now, there is no Laman type theorem for 3-dimensional generic rigidity and there is no combinatorial characterization for $3D$ geometric constraint problems; it's an open problem.

2.9 Other application of geometric constraints solving

In this section, we present some other important applications of geometric constraints solving: Localization in Wireless Sensor Networks, molecular biology, and dynamic geometry.

2.9.1 Localisation in Wireless Sensor Networks

In many applications for wireless sensor networks (WSN), the geographic information can be critical. The association of the location information with any other type of information may be used by routing algorithms, meteorological information, agricultural, military applications, etc. A WSN can be modeled as a graph, distances between nodes are estimated using physical characteristics of the communication networks, like signal strength. The idea of these algorithms is to estimate the coordinates of mobile or fixed unknown nodes based on their connectivity. Most localization methods are based on the idea that some nodes (called anchor) know their coordinates (e.g., GPS-equipped nodes or specified). Anchors transmit their coordinates to help other nodes to localize themselves. When unknown nodes deduce its location, it becomes a new anchor (see figure 2.18). Rapidly, unknown nodes can estimate their coordinates. For more details, see for example Mao et al. [31] or Zhang et al. [3]. Rigidity gives the theoretical base of most localization algorithms.

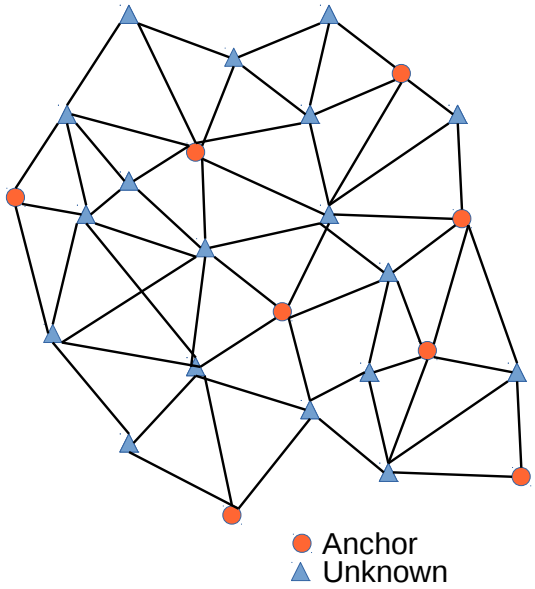


FIGURE 2.18: Example of WSN topology.

2.9.2 Molecular Biology

The protein structural flexibility and dynamics is as critical for protein function as its 3D-structure [32]. There is a direct link between protein's function and its three-dimensional structure. Detecting the flexible and the rigid parts of the protein can be very helpful in the study of some diseases. Nuclear magnetic resonance (NMR) experiments on protein can be used to determine a subset of distances between atoms in protein's molecules. Hence, the problem can be translated into constraint representation of the protein, i.e., a geometric constraint graph, which will be analyzed using most rigidity theory results. See [33], [34] and [32] for more details. The figure 2.19 is taken from [32], it represents the rigidity analysis of a protein. It was generated using a software called "FIRST" developed by Sljoka [32].

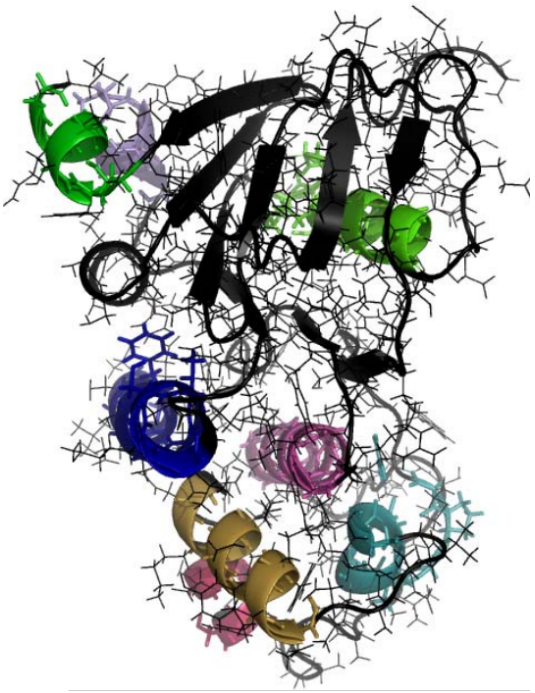


FIGURE 2.19: A Rigidity decomposition of a protein

Each coloured region represents a single rigid rigid body where all bonds in each cluster are non-rotatable and can only move using trivial rigid body motions (i.e., translations and rotations). The connecting gray regions are flexible.

2.9.3 Dynamic Geometry

Dynamic geometry is a new way of exploring classical geometry using interactive computer software [35]. The computer can record the way the user constructs and places the geometric elements; the construction will be very quickly rebuilt if the user changes the values assigned to some parameters. This leads to an easy understanding of the dynamic behaviour of the geometric construction when some of its distances are modified. In such software, the user can create geometric constructions and manipulate them interactively. Construction is composed of many geometric elements such as points, lines, and conics whose positions have been constrained in different ways. Dynamic geometry software have been used in high-school geometry teaching [36], with

a focus on classical, two-dimensional Euclidean geometry. The research laboratory in applied sciences, image Processing, Computer Vision and Robotics can also use dynamic geometry in many applications such as simulation. *GeoGebra* [37] for example is a well-known software that can be used at the college level as well as in a research laboratory. Figure 2.20 gives a GeoGebra software screenshot. In such systems, when the user sketches a construction, he may require, for example, that a line must be tangent to a certain conic, the role of the system is to make sure that this constraint is always satisfied when the construction is updated. The computer can record how the geometry was built by a user which allows it to quickly reconstruct the figure whenever the user modifies the values assigned to some parameters. The user defines geometric elements with some relationships between them (constraints), then he can explore the dynamic behaviour of the remaining geometric objects when one of them is moved. In [38], Freixas et al. apply some well-known results from the Geometric Constraint Solving field, to unambiguously capture the expected dynamic behaviour of a given geometric problem.

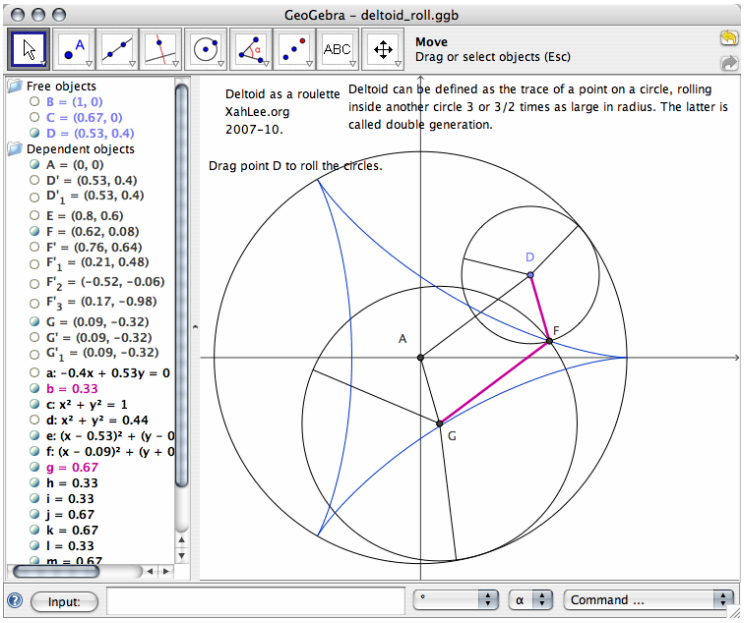


FIGURE 2.20: A GeoGebra screenshot.

Chapter 3

Major Solving Approaches in CAD

Contents

3.1 Numerical Solvers 36

3.1.1 Newton-Raphson method 36

3.1.2 Bisection method 37

3.1.3 Homotopy method 39

3.2 Other solving Approaches 40

3.2.1 Degrees of Freedom Analysis Approaches 40

3.2.2 Propagation Approaches 40

3.2.3 Logic-Based Approaches 41

3.2.4 Symbolic Approaches 41

3.3 Conclusion 43

3.1 Numerical Solvers

Numerical solvers were one of the first methods for geometric constraint solving. The GCS is translated into a system of algebraic equations, which have zero value when the constraints are satisfied. The resulting system of equations is solved with iterative algebraic methods, such as Newton-Raphson, the bisection method, homotopy or others. All those methods can solve general nonlinear systems of algebraic equations, but may require at worse $O(n^3)$ running times. In the following, we give a brief description of some uses of numerical solvers in geometric constraint solving. Chapter 4 presents graph based solver where the numeric resolution will be accelerated by the decomposition.

3.1.1 Newton-Raphson method

Newton-Raphson method is a popular and powerful iterative method for solving system of non-linear equations. In n dimension, it has the following form:

$$x_{k+1} = x_k - J_f(x_k)^{-1} f(x_k) \tag{3.1}$$

where $J_f(x)$ is the Jacobian matrix of f,

$$\{J_f(x)\}_{ij} = \frac{\partial f_i(x)}{\partial x_j} \tag{3.2}$$

To save computational time, $J_f(x_k)$ will not be inverted, but instead we solve the linear system:

$$J_f(x_k)s_k = -f(x_k) \tag{3.3}$$

the next iterate that will be taken is:

$$x_{k+1} = x_k + s_k$$

After a GCS is transformed into a system of equations, three unknown coordinates of two vertices of a randomly chosen edge will be fixed (that correspond to fixing two geometric elements in the Euclidean space). The initial approximative coordinates will be guessed; we can use for example the coordinates of the geometric elements, specified by the user, in the initial sketch. The iterating process is then started and repeated until converging to an accurate solution. The linear system of the equation 3.3 is solved using the LUP decomposition, which is faster and more accurate than Gaussian elimination method. Another alternative is the Krylov space methods [39]. An interest of these methods is that they work even for an irreducible system: they only use the sparsity of the Jacobian matrix. For further details see for example [40]. Because the computation of the Jacobian is a difficult and expensive operation, another alternative to Newton-Raphson method is the Broyden's method. This method computes the Jacobian matrix only at the first iteration (see [41] page 214 for more detail).

3.1.2 Bisection method

Newton method has the following two major drawbacks [42, 43]: In general case, Newton's method does not converge or converges to an unwanted solution; it requires an initial approximation of the solution that can be provided interactively by the user. Those two problems make the use of this method

very limited in geometric constraint solving. In [42], Ait-Aoudia et al. proposed the use of the bisection method [44] to solve the non-linear system of equations that arise from GCS. Let $F(X)$ be a system of non-linear equations:

$$F(X) : f_i(x_1, x_2, ..., x_n) = 0; 1 \leq i \leq n; a_i \leq x_i \leq b_i$$

where a_i and b_i are known bounds of the variables x_i. The solving process of the system $F(X)$ consists of a binary search for solutions in the interval vector (an n-dimensional rectangle called box) $(X_1, X_2, ..., X_n)$, where $X_i = [a_i, b_i]$ and $x_i \in X_i$. To find all the solutions of such system, Moore and Qi proposed in [45] a successive interval test for existence and uniqueness of a solution to a non-linear system and for convergence of iterative methods. The searching process is a divide-and-conquer approach, a cyclic sequence of bisections of X. Let $X^{(0)}$ be the initial box where $f_i(X^{(0)})$ may contain a solution of $F(X)$ for $1 \leq i \leq n$. Bisect $X^{(0)}$ in coordinate direction x_1, i.e., the box $X^{(0)}$ is divided in half. We have now two new boxes, each one may contain a solution. Exclude any box for which $0 \notin f_i(X^{(0)})$ for some i. Bisect in coordinate x_2 the new boxes that may contain a solution. Repeat the process until an interval is identified and an iterative method is used to find the root or no solution exists.. The operation is recursively repeated until all the solution are found. Figure 3.1 shows a two-dimensional illustration of the process of searching for the root of a system of equations: $f_1(x_1, x_2) = 0$ and $f_2(x_1, x_2) = 0$. The regions in between are marked $++, +-, -+$ or $--$, according to the signs of f_1 and f_2 respectively in those regions. Bisection method is safe, always converges, but it is slow. Newton gains in speed but may not converge [46]. To speed-up the searching process Ait-Aoudia and Mana proposed in [47] a distributed implementation of the bisection.

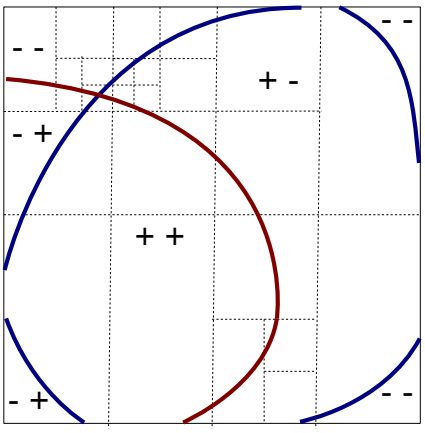


FIGURE 3.1: A two-dimensional illustration of the bisection method.

3.1.3 Homotopy method

The resolution by homotopy is a standard method of numerical analysis. The use of this method for geometric constraint solving was first proposed by Lamure and Michelucci in [48]. In 2D CAD software, the user first defines a sketch; then the system corrects the sketched model by solving its corresponding GCS. The homotopy method works in a similar manner. Let $F(X)$ be the system of equations to be solved. $F(X) : f_i(x_1, x_2, ..., x_n) = 0; 1 \leq i \leq n$. Suppose that we have another system of equations: $G(X) : g_i(x_1, x_2, ..., x_n) = 0; 1 \leq i \leq n$ with a known solution $S = (s_1, s_2, ..., s_n)$. S represents the initial guess interactively provided by the user, i.e., the coordinate of the geometric objects defined in the sketched model.

$$G(X) = F(X) - F(S)$$

F and G are then embedded in a homotopy:

$$H(t, X) = tF(X) + (1 - t)G(X)$$

such that:

$$H(0, X) = G(X) \text{ and } H(1, X) = F(X)$$

The system H is a linear interpolation between G and F. The main idea of

resolution by homotopy is to follow the homotopy curve, starting from $t = 0$, $X = S$. If the homotopy curve passes through a point (t, X) with $t = 1$, then a solution to $H(1, X) = 0$, and thus to $G(X) = 0$ has been found. Because a single path is followed, one solution which is closer to the specified sketch by the user is found. A detailed study on the use of homotopy method in geometric constraint solving can be found in [49]. We note that damped Newton [50] can be used as an alternative to the homotopy.

3.2 Other solving Approaches

3.2.1 Degrees of Freedom Analysis Approaches

In DOF analysis approaches, the solver reduces the system's remaining DOFs by incrementally satisfying each constraint. An example of such solver is proposed by Kramer in [51, 52]. The GCS is transformed into a graph where vertices are labelled with the number of degrees of freedom of the represented geometric object. Each graph edge is marked by the degrees of freedom cancelled by the designated constraints. For example, if the incident vertices are points in 2D, a distance constraint cancels 1 degree of freedom. The algorithm analyses the graph to cancel all possible DOFs. Hsu and Brüderlin proposed in [53] a hybrid method that starts by applying geometric constructions whatever possible. Iterative methods are used when the geometric construction is not possible.

3.2.2 Propagation Approaches

In Propagation approaches, the GCS is transformed into a set of algebraic equations, which are represented by a symmetric graph, vertices are variables

and equations. Edges are labelled with the occurrences of the variables in the equations. The algorithm is an orientation of the edges in the graph in such a way that each equation vertex is a sink for all the edges incident on it except one. If the process succeeds, then there is a general incremental solution. Freeman-Benson et al. presents an incremental constraint solver, called DeltaBlue, that maintains a solution to the constraint hierarchy as constraints are added and removed. The constraint hierarchy is created by separating constraints into levels according to their relative importance. Veltkamp and Arbab presented in [55] an incremental approach to geometric constraint satisfaction that categorises solution into what they called quanta. In [56], Borning et al. study the inequality constraints that are useful for specifying various aspects of user interfaces.

3.2.3 Logic-Based Approaches

In such solving methods, the GCS is transformed into a set of geometric assertions and axioms. Aldefeld[57], Brüderlin[58], and Yamaguchi and Kimura[59], use first order logic to derive geometric information applying a set of axioms from Hilbert's geometry. Sunde[60] proposed a system architecture that combines a production mechanism and a verification mechanism. Verroust et al.[61] proposed an approach based on the use of an expert system to uncouple constraint equations, and to find a possible sequence for the computation of the geometric elements. Joan-Arinyo and Soto [62] formalize the solver as a rewrite system whose rewriting rules are geometric construction steps.

3.2.4 Symbolic Approaches

A GCS is transformed into a nonlinear system of equations, which will be solved symbolically, using for example Buchberger algorithm. Buchberger [63]

uses the Gröbner bases to triangularize the system, Buchanan and de Pennington[64] report a solver built on top of the Buchberger's algorithm.

This is a rewriting algorithm, which considers each equation as a rule: the most significant monomial is rewritten in contrast to the rest of the polynomial. For example, $x^2 - 2 = 0$ gives the rule: $x^2 \rightarrow 2$ where $\rightarrow$ means "is rewritten to". Buchberger algorithm ensures that the system of rules is confluent, i.e., the result of the expression rewriting does not depend on the order in which the rules are applied.

For this, the algorithm considers the critical pairs: these are the simplest monomials that can be rewritten at least two ways. For example, if a monomial gives xy (non-reducible) or 2 according to the first applied rule, then, the rule $xy \rightarrow 2$ is added to the set of the rules. With two rules: $x^2 \rightarrow 2$ and $xy \rightarrow 2$, the simplest monomial that can be rewritten in two ways is: $x^2 \rightarrow y$, is rewritten by either $2x$ or $2y$. We deduce $2x - 2y = 0$ and therefore the rule $x \rightarrow y$ or $y \rightarrow x$, depending on the order that we have fixed on the variables and monomials. The most common orders are: lexicographical order, reverse lexicographical, or the total order then lexicographical, or the total lexicographical then reverse lexicographical. Basically, if the monomial $x^d = x_1^{d_1} x_2^{d_2} ... x_n^{d_n}$ is less than $x^e = x_1^{e_1} x_2^{e_2} ... x_n^{e_n}$, then $x^c x^d$ must also be less than $x^c x^e$. The theory shows that the algorithm terminates after a finite number of steps. However, the number of steps can be exponential or doubly exponential. When the algorithm terminates, the rule system is confluent, and the two terms A and B are equal if and only if the result of rewriting $A - B$ is the identically zero polynomial, 0, or if the introduction of the rule $(A - B)\alpha - 1 = 0$ (where α is a new variable) causes a contradiction: the insertion of the rule $1 \rightarrow 0$.

In 2D space, for example, each equation involves the coordinates x_i, y_i and x_j, y_j of two vertices i and j. The idea is to pick one given equations and

solve for either of the variables, x or y. The result in the first step will be substituted into the other equation. This process will be repeated until a single equation with one variable which can be solved as usual. A lexicographical or inverse-lexicographical order must be used. In the end, we deal with a simple equation to solve.

Kondo [65] and Borning [66] report a symbolic algebraic method. The main drawback of such solvers is the computational complexity which is exponential; this leads to their usefulness in constraint-based CAD applications.

3.3 Conclusion

Logic based solvers, symbolic methods or propagation methods are exponentials (or doubly exponentials) in time, their use is declined in the recent years. Numerical solvers are $O(n^3)$ or worse; any acceleration is welcome. The next chapter is dedicated to the graph based solvers, where the GCS problem is decomposed into smaller sub-problems. The goal is to limit the use of numerical solvers to the smallest possible system of equations.

Chapter 4

2D Graph-based Solvers

Contents

4.1 Introduction **45**

4.2 2D geometric constraint graph and their decomposition **46**

4.3 Desirable requirements of a graph-based solver . **47**

4.4 Constructive Solvers **48**

4.4.1 Decomposition analysis method 48

4.4.2 Reduction analysis method 49

4.4.3 Tree Decomposition Method 53

4.5 General Solvers **55**

4.5.1 Geometric Constraint Bipartite Network 55

4.5.2 Detection of solvable subgraphs 56

4.5.3 Maximum Matching Approach 59

4.5.4 Some remarks on the MM Algorithm 64

4.5.5 Condensing and Frontier algorithms 65

4.5.6 Recursive Skeletonization Algorithm 68

4.6 Conclusion **72**

4.1 Introduction

Among the major 2D geometric constraint solving approaches are those based on the structural analysis of the graph representation of the GCS. Graph-based solvers have become the dominant class of geometric constraint solvers [7]. The geometric constraint system is transformed into a graph; the vertices represent the geometric elements, and the edges represent the specified constraints between them. The solver analyses the structure of the graph and generates a plan of resolution (also called a DR-plan).

In this work, we focus on 2D graph-based solvers, developed in Computer-Aided Design context. Many such solvers transform the GCS into a graph. By applying some decomposition techniques on the constraint graph, they isolate under, over, and well-constrained parts. The well-constrained part is then analyzed by a decomposition technique to find small, solvable subgraphs. The final solution is produced by merging the solved subgraphs in respect to an order of resolution produced in the decomposition phase. This is referred to as Decomposition/Recomposition plan (DR-Plan) [15]. The primary aim of this decomposition is to speed up the resolution process by limiting the use of direct algebraic resolution to subsystems that are as small as possible (typically ruler and compass solvable problems).

2D graph-based solvers are classified by Hoffman et al. [15] in two main classes: constructive solvers (called also SR-Planners, constraint shape recognition), and general solvers (MM-planners generalized maximum matching). Those two types of solvers differ especially in the domain of geometric constraints problems. Examples of constructive solvers are those proposed by Joan-Arinyo et al. [16], Fudos and Hoffmann [11], Hoffmann and Vermeer [67], Ait-Aoudia

et al. [68], Bouma et al.[69] and Owen[14]. The second categories is called MM-planners, like those proposed by Ait-Aoudia et al. [10], Hoffman et al.[12], Ait-Aoudia and Foufou[9], Latham and Middleditch[13] and Kramer[52]. Their principle is based on finding solvable subgraphs by transforming the constraint graph into a bipartite graph and finding a maximum generalized matching, followed by a connectivity analysis to obtain the DR-plan. Figure 4.1b gives a graph that can be solved by any constructive solver, the graph of figure 4.1a is solvable only by MM-Planners.

(a) (b)

FIGURE 4.1: Two well-constrained graphs: (a) solvable by SR-Planners and (b) solvable only by MM-Planners

4.2 2D geometric constraint graph and their decomposition

Any GCS can be represented by a graph $G(V, E)$, which consists of a vertex set V and an edge set E. The vertices of G represent the geometric elements, and the edges represent the constraints between them. The cardinality of V will be called the size of G. Laman's theorem [18] gives the necessary condition of generic solvability for any GCS. To be solvable, its constraint graph must be structurally well-constrained, (also called generically isostatic, minimally rigid or Laman graph by rigidity theory and structural topology communities).

Definition 4.1. A geometric constraint graph $G = (V, E)$ where $|V| = n(n > 1)$ and $|E| = m$ is structurally well-constrained if and only if $m = 2n - 3$ and

$m' \leq 2n' - 3$ for any induced subgraph $G' = (V', E')$, where $|V'| = n'(n' > 1)$ and $|E'| = m'$.

Definition 4.2. A constraint graph $G = (V, E)$ contains a structurally over-constrained part if there is an induced subgraph $G' = (V', E')$ having more than $2n' - 3$ edges.

Definition 4.3. A geometric constraint graph $G = (V, E)$ is structurally under-constrained if it is not over-constrained and the number of edges is less than $2n - 3$.

Laman's theorem [18] is proved only for point to point distance constraints. There is no combinatorial characterization for any other geometric element [14]. The extension of his theorem to other geometric elements may imply incorrect cases. Typical examples are given in section 2.7.

4.3 Desirable requirements of a graph-based solver

Graph-based solvers can be compared using many desirable properties as defined below. We note that those properties can be contradictory, the satisfaction of one of them can be to the detriment of another.

- *Generality*: this property defines the domain of GCS and characterizes the possible geometric configurations handled by the solver: a solver is general if it can decompose any possible configuration.
- *Optimality*: the size of the largest detected subgraph has to be close to minimal as proposed by [15] and the number of identified subgraphs have to be maximal [1].

- *Computational complexity*: The solver has to be fast in practice, allowing an interactive use.
- *Handling over- and under-constrained parts*: the solver has to isolate under- and over-constrained subgraphs for further correction.

4.4 Constructive Solvers

Examples of constructive solvers (called also SR-Planners) are given in [11, 14, 16]. These solvers are not complete, i.e., they do not solve every well-constrained graph but only a sub-class of ruler and compass constructible geometric configurations. These methods require that the constraint graph is biconnected [16], and the smallest solvable subgraphs are triangles [15]. Those algorithms can be theoretically extended to handle other types of shapes than triangles by giving an explicit recognition algorithm to each new kind of shapes. We note that there is an infinite collection of shapes [17]. As a result, SR-planners cannot be generalized. In the construction phase, constructive solvers deal with only quadratic equations. Hence, the calculation of coordinate does not require complex mathematical computation. Next, we will detail three well known constructive solvers.

4.4.1 Decomposition analysis method

Owen [14] presented a graph based approach that solves a sub-class of geometric constraint systems that are constructible by ruler-and-compass. This method is considered as the first graph-based approach to be proposed in the literature. It is composed of two phases. The first one, called decomposition analysis phase, is based on the analysis of the geometric constraint graph by finding all the bi-connected components (see Algorithm 2). It is a divide and

conquers algorithm that recursively breaks down the GCG into two subgraphs at each step. This is done by recursively splitting the graph in what he called an articulation pair. An articulation pair is composed of two nodes whose the removal causes the division of the graph into two new graphs. The splitting process is repeated until the graph becomes simple enough to be solved directly, i.e., triangles or simple edges. To preserve the rigidity of the graph, virtual edges are added as necessary. Virtual edges are removed next for allowing further decomposition. The complexity of this algorithm is $O(n^2)$. Figure 4.2 shows an example of such decomposition. Figure 4.2a represents the constraint graph, this graph is split at the articulation pair (H, D) (Figure 4.2b); a virtual edge is added; similar splitting is done to the other articulation pairs (E, D) and (D, H) (Figures 4.2c, 4.2d). Others splitting are done without adding virtual edges because one of the component is a single edge (Figure 4.2d, 4.2e, 4.2f, 4.2g).

The second phase of Owen's method is a combination of solved subgraphs in a reverse order of the analysis phase. At each combination, translation and rotation in $2D$ space are necessary (using only quadratic equations). If this algorithm deals with tri-connected components, the GCS can't be solved by Owen's method because the equations cannot be solved quadratically. [14].

4.4.2 Reduction analysis method

In [11], Fudos and Hoffmann presented an $O(n^2)$ algorithm that can solve the same class of geometric constraint problems as Owen's algorithm[14]. The algorithm is composed of two phases. The first one, called analysis phase, where the reduction algorithm of Fudos et al [11] is as follow: let $G(V, E)$ be the geometric constraint graph to be analysed. The first step of the algorithm corresponds to the construction of the initial set of clusters

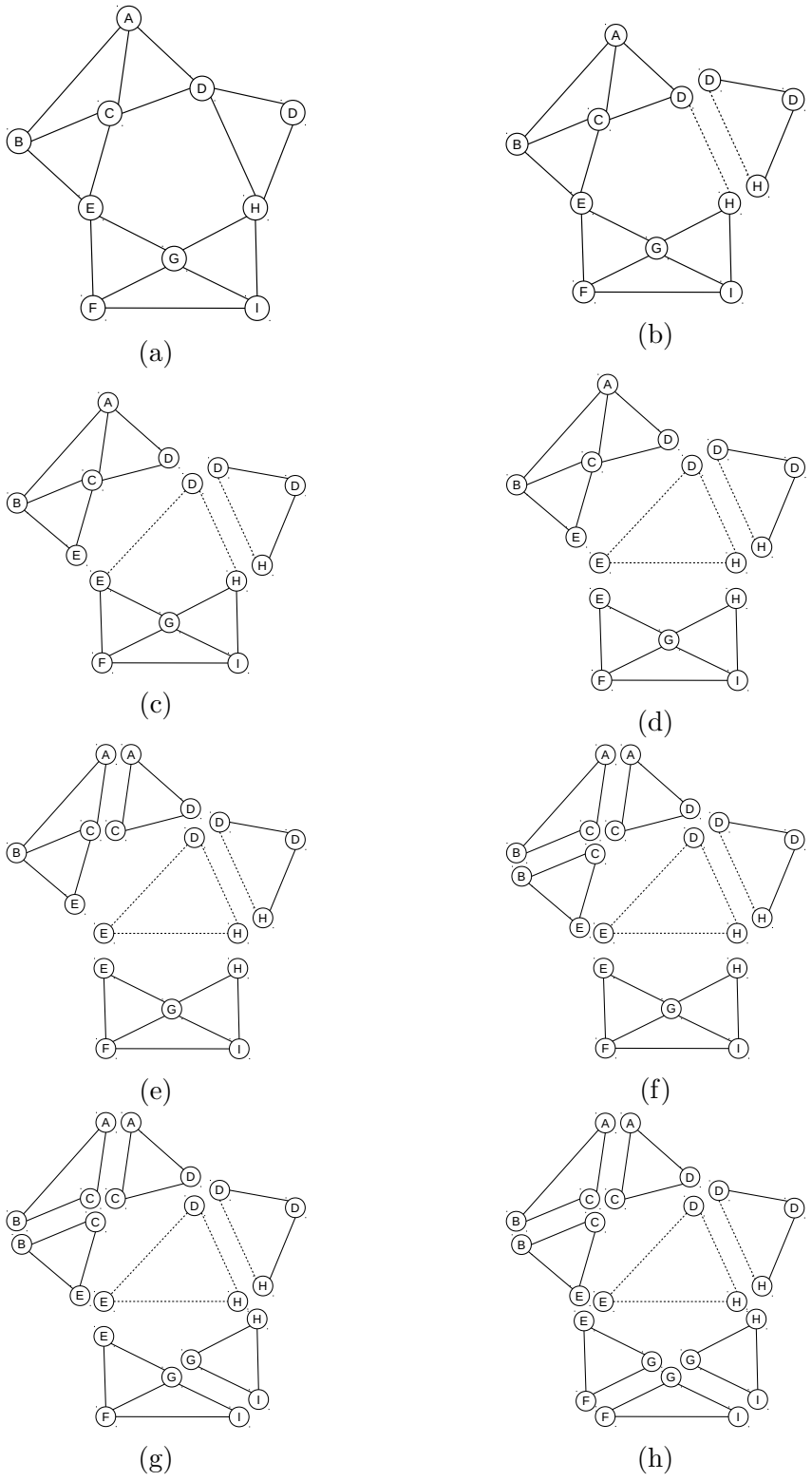


FIGURE 4.2: A geometric constraint graph decomposition by Owen's algorithm

Algorithm 2 Owen's decomposition algorithm

Input: G: the geometric constraint graph;

Output: A plan of resolution.

1. Split the Geometric constraint graph into bi-connected components.
2. Split each bi-connected component into split components, inserting virtual edges as necessary.
3. At any articulation pair with no single edge and exactly one more complex subgraph, remove the virtual edge from the subgraph. Apply the algorithm recursively from stage 2.
4. The algorithm terminates when the graph cannot be split further. '

$S_G = \{\{u, v\}/(u, v) \in E\}$. Each cluster represents a basic rigid body. The second step of the algorithm is a recursive merging of each three clusters C_1, C_2, and C_3 such that: $\exists g_1, g_2, g_3 \in V, C_1 \cap C_2 = \{g_1\}; C_2 \cap C_3 = \{g_2\}; C_1 \cap C_3 = \{g_3\}$ and $g_1 \neq g_2 \neq g_3$. (i.e., those tree clusters pairwise intersect in a singleton). Geometrically this merging corresponds to the computation of the relative coordinates of three geometric objects. Its implantation uses a bipartite graph $H(V_H, E_H)$, where $V_H = Y_H \cup X_H$ and $Y_H \cap X_H = \emptyset$. Each element of Y_H corresponds to a cluster in S_G, X_H is the set of geometric elements. The clusters merging corresponds to the finding of 6-cycle in the cluster graph H, then rewriting them as shown in 4.3. This process is recursively repeated until the graph H is rewritten to a star. The cluster merging is recorded, it represents the resolution plan, it will be used as input for the construction step.

The following is the reduction of the graph shown in figure 4.2a. Merged clusters are represented in bold font.

$\mathbf{\{\{A,B\}}, \mathbf{\{A,C\}}, \{A, D\}, \mathbf{\{B,C\}}, \{B, E\}, \{C, D\}, \{C, E\}, \{D, H\}, \{D, J\}, \{E, F\}, \{E, G\}, \{F, G\}, \{F, I\}, \{G, H\}, \{G, I\}, \{H, I\}, \{H, J\}\} \rightarrow$

$\{\{A, B, C\}, \{A, D\}, \{B, E\}, \{C, D\}, \{C, E\}, \mathbf{\{D,H\}}, \mathbf{\{D,J\}}, \{E, F\}, \{E, G\}, \{F, G\}, \{F, I\}, \{G, H\}, \{G, I\},$

Algorithm 3 Reduction analysis algorithm

Input: a geometric constraint graph G;

Output: a plan of resolution.

procedure REDUCE(G)

1. Construct initial set of clusters S_G, each element of S_G is a set of two adjacent vertices of G.

2. Construct the cluster bipartite graph H.

3. Find all triangles in G.

4. Successively rewrite H by replacing a 6 cycle in H by a four node structure as explained in figure 4.3. Record a cluster merging operation for each such rewriting step

5. If H can be rewritten into a final graph that is a star with center a cluster and periphery the vertices of G, then G is solvable; otherwise, it is not solvable.

end procedure

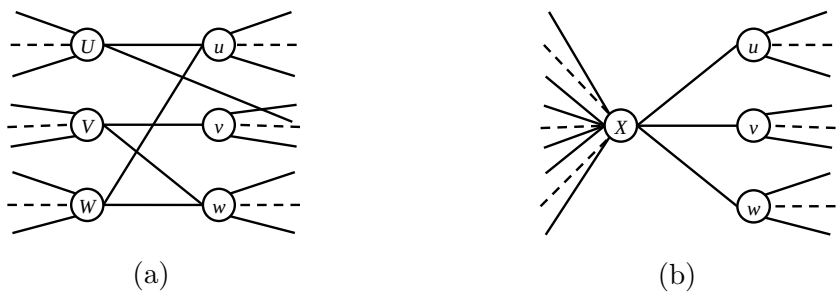


FIGURE 4.3: Reduction of a 6-cycle of the bipartite graph H that corresponds to a cluster merging.

$\{H, I\}, \mathbf{\{H,J\}}\} \rightarrow$

$\{\mathbf{\{A,B,C\}}, \mathbf{\{A,D\}}, \{B, E\}, \mathbf{\{C,D\}}, \{C, E\}, \{D, H, J\}, \{E, F\}, \{E, G\}, \{F, G\}, \{F, I\}, \{G, H\}, \{G, I\}, \{H, I\}, \} \rightarrow$

$\{\mathbf{\{A,B,C,D\}}, \mathbf{\{B,E\}}, \mathbf{\{C,E\}}, \{D, H, J\}, \{E, F\}, \{E, G\}, \{F, G\}, \{F, I\}, \{G, H\}, \{G, I\}, \{H, I\}, \} \rightarrow$

$\{\mathbf{\{A,B,C,D,E\}}, \{D, H, J\}, \{E, F\}, \{E, G\}, \mathbf{\{F,G\}}, \mathbf{\{F,I\}}, \{G, H\}, \mathbf{\{G,I\}}, \{H, I\}, \} \rightarrow$

$\{\mathbf{\{A,B,C,D,E\}}, \{D, H, J\}, \{E, F\}, \{E, G\}, \mathbf{\{F,G,I\}}, \mathbf{\{G,H\}}, \ \mathbf{\{H,I\}}, \} \rightarrow$

$\{\{A, B, C, D, E\}, \{D, H, J\}, \{E, F\}, \{E, G\}, \{F, G, I, H\}, \} \rightarrow$

$\{\{A, B, C, D, E\}, \{D, H, J\}, \mathbf{\{E,F\}}, \mathbf{\{E,G\}}, \mathbf{\{F,G,I,H\}}, \} \rightarrow$

$\{\mathbf{\{A,B,C,D,E\}}, \mathbf{\{D,H,J\}}, \mathbf{\{F,G,I,H,E\}}, \} \rightarrow$

$\{\{A, B, C, D, E, F, G, H, I, J\}\}$

4.4.3 Tree Decomposition Method

In [16], Joan-Arinyo et al. presented a new $O(n^2)$ constructive solver that belongs to the constraint shape recognition category, i.e., it solves the same domain of geometric constraints systems as Fudos's [11] and Owen's [14] solvers. The algorithm decomposes biconnected geometric constraint graphs by searching for hinges in fundamental circuits of a specific planar embedding of the constraint graph. A planar graph is a graph that can be embedded in the plane, i.e., it can be drawn in such a way that no edges intersect with each other. The algorithm recursively decomposes the problem into three subproblems that pairwise share a common geometric primitive, called hinge. The algorithm has two main steps:

1. Transform the GCG into a simpler, planar graph and
2. Compute a planar embedding for the transformed graph where hinges are identified as a set of three vertices shared by two faces.

The algorithm is based on the decomposition of a graph $G = (V, E)$, according to the bridges induced in G by a circuit C. It is inspired by the algorithm for finding the tri-connected components of a graph. If $G = (V, E)$ is the geometric constraint graph; the algorithm starts by computing a spanning tree T for G using a depth-first search and computing the set of fundamental circuits induced by the spanning tree. Then, the algorithm seeks for a fundamental circuit C with a set of hinges that defines a set decomposition of V. If the

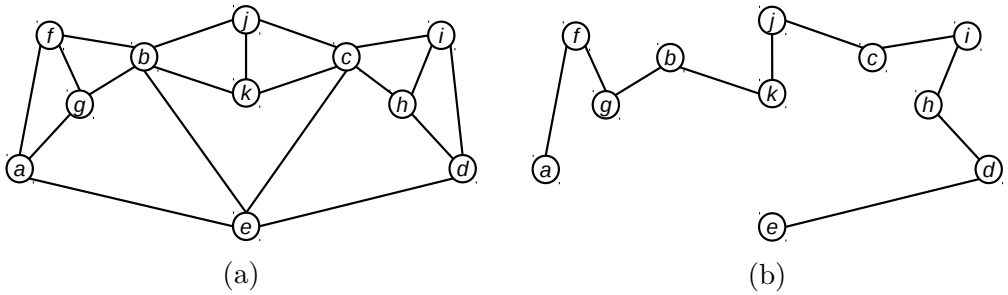


FIGURE 4.4: A geometric constraint graph and its spanning tree.

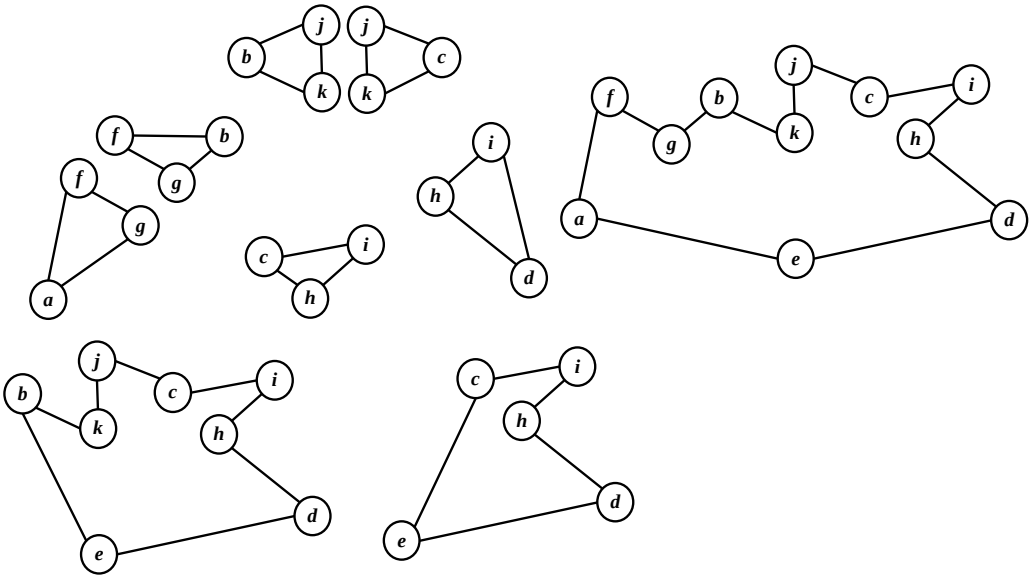


FIGURE 4.5: Set of fundamental circuits of graph 4.4a.

algorithm fails in finding a fundamental circuit with a set of hinges, the input graph is not tree-decomposable.

Figure 4.4a gives a geometric constraint graph, figure 4.4b shows the resulting spanning tree. The set of fundamental circuits induced by the spanning tree considered are shown in figure 4.5. Every chord in the spanning tree induces a fundamental circuit. We recall that a chord is an edge $(e_i, e_j) \in V$ such that $(e_i, e_j) \notin T$

4.5 General Solvers

In the class of general solvers [9, 10, 12, 70]) there is no restriction on the domain of the 2D geometric constraint configurations, the smallest subgraph can be any irreducible well-constrained graph. The most important building block of those methods is the irreducible subgraphs detection procedure. These categories use a maximum matching or a flow based algorithm to detect small, solvable subgraphs and their order of resolution. Those methods are complete, i.e., any well-constrained graph can be decomposed, solved and recomposed. Next, we present an Algorithm proposed by Hoffmann et al. [71] for the detection of all solvable subgraphs (called also dense subgraphs)..

4.5.1 Geometric Constraint Bipartite Network

In 2 dimensions, every geometric element has 2 degrees of freedom and all constraints between them are binary and eliminate one degree of freedom. Thus, in the corresponding constraint graph, the weight of all the edges is 1 and of all the vertices is 2. When the dimension space is more than 2, each vertex $v \in V$ has a weight $w(v)$ that represents the DOF of its corresponding geometric element. Each edge $e \in E$ has a weight $w(e)$ that corresponds to the number of DOFs that are eliminated by its corresponding constraint. In this case, a GCS is represented by a weighted constraint graph $G(V, E, w)$. To find irreducible solvable subgraphs, general solvers transform the constraint graph $G(V, E, w)$ of a GCS to a bipartite directed network $G^* = (M, N, s, t, E^*, w)$ where:

1. s is the source and t is the sink.
2. The vertices in N are the vertices in V.
3. The vertices in M are the edges in E.

4. The edges of G^* are $E^* = \{(e,u)(e,v)|e = (u,v), e \in E\}$.

5. Every directed edge (s,e) from the source node s to the set M is weighted by $w(e)$.

6. Each edge (v,t) from N to the source node t is weighted by $w(v)$.

7. Each edge (v,t) from N to the source node t is weighted by $w(v)$.

8. The edges from N to M are weighted by ∞.

Figure 4.6 gives an example of a 2D geometric constraint graph and its bipartite network. Many algorithms use the constraint bipartite network for GCS decomposition. Ait-Aoudia et al. [10] use this approach to find all the irreducible with their order of resolution. Hoffmann et al. [71] also use this approach to find all dense subgraphs. The next section gives more details on those two approaches.

4.5.2 Detection of solvable subgraphs

In [71], Hoffmann and Vermeer presented an $O(n(m+n)))$ algorithm that isolates solvable subgraphs (called also dense subgraph) from a constraint graph $G(V,E)$ where $|V| = n$ and $|E| = m$. The proposed algorithm is general in the sense of handling the cases where the DOFs of geometric elements can be greater than two and a constraint can remove more than one DOF. This algorithm is flow-based and generalizes prior maximum matching approaches as those proposed by Ait-Aoudia et al. [10]. The principle of finding dense subgraphs is based on the definition below [71].

Definition 4.4. Let $G(V,E)$ be a constraint graph of GCS. An induced subgraph $A(V_A, E_A) \subseteq G$ is called dense if $|E_A| - 2|V_A| > K$, where K is a constant.

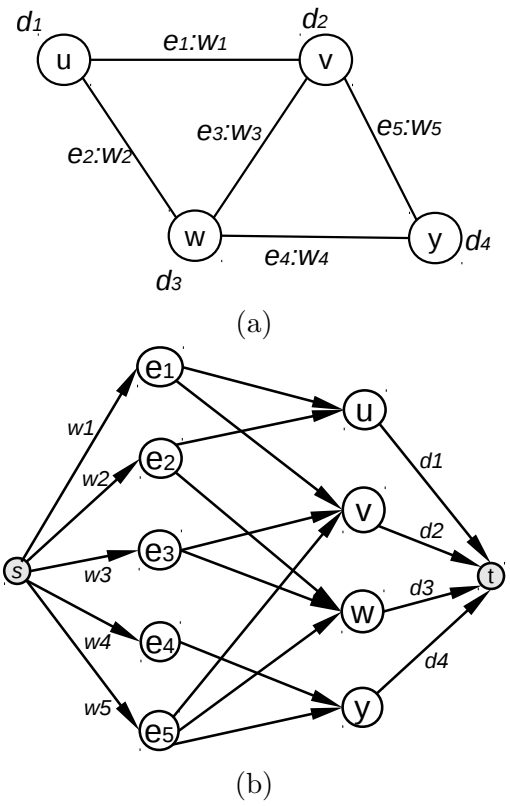


FIGURE 4.6: (a) A constraint graph, (b) its bipartite associated network

A dense subgraph represents an irreducible sub-problem of the GCS. The goal is to find a *minimal* dense subgraph, that is, a dense subgraph A such that A does not contain a proper dense subgraph B. The goal is restricted to minimal dense subgraph (i.e. for inclusion) and not *minimum* because finding the minimum dense subgraph B is shown to be NP-hard [71].

For geometric constraint problems, the constant $K = -(D + 1)$ where D represents the number of DOFs that a rigid figure can have. In $2D$ space, $D = 3$ and in $3D$ space, $D = 6$. If the figure is to be fixed with respect to a global coordinate system, then $D = 0$ is required. Let's explain the principle of this algorithm using the figure 4.7. Figure 4.7a represent a GCS graph composed of three geometric elements, having different DOFs values as showed by their weight. As represented by edges weight, every constraints can eliminate 3 DOFs. The associated network is given by figure 4.7b. Figures

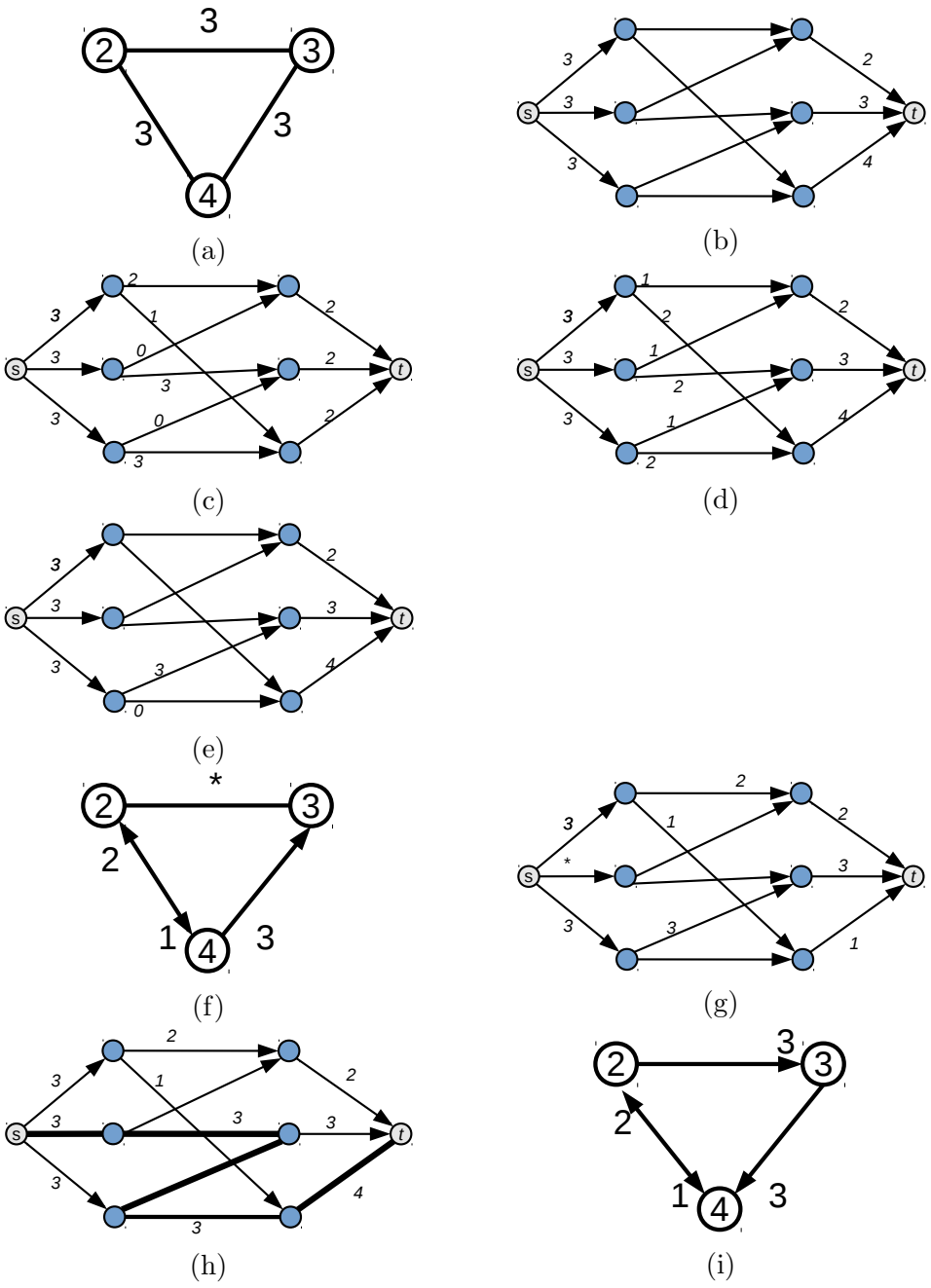


FIGURE 4.7: (a)A flow for the constraint graph (b) another flow, (c) the initial flow

4.7c and 4.7d show different flows for the constraint graph. To find the dense subgraph in a faster way, this algorithm uses a tailored version of the general maximum flow algorithm. It starts with the empty subgraph G' of G and adds to it one vertex v at a time. When a vertex v is added, it considers the adjacent edges e incident to G'. For each e, try to distribute the weight $w(e)$ to one or both of its endpoints without exceeding their weights. If we can distribute all edges, then G' is not dense. If there is a dense subgraph, then there is an edge whose weight cannot be distributed even with redistribution. Distributing an edge e in G now corresponds to pushing flow equal to the capacity of (s, e) from s to t in G^* (see figure 4.7f, 4.7g, 4.7h, 4.7i)

4.5.3 Maximum Matching Approach

The maximum matching approach was first used by Serrano [72] for systems of non-linear equations appearing in conceptual design problems. Ait-Aoudia et al. [10] proposed the first general solver that detects all the subgraphs with their order of resolution. We have successfully implemented this algorithm to test our generator presented in chapter 5. This algorithm can also isolate the under- and over-constrained subgraphs. It is based on the calculation of the maximum matching of the bipartite graph $G(M, N, E)$ associated to the GCS. For a the geometric constraints $G(V, E)$ The bipartite constraint graph $G^B = (M, N, E^B)$ is constructed as follow:

First, we pin an edge (u, w) i.e., we consider that the coordinates of geometric elements that corresponds to u and w are fixed in the plane.

The vertex set M represents the DOFs of the unpinned nodes. For each node $v_i \in V$, we create k nodes copies: $v_i^1, .., v_i^k$, such that k is equal to the DOFs number of v_i ($k = 3$ if we are in 2D space). For each edge $e = (v_l, v_m) \in G$ we create a node $e_N \in N$ and we add edges (v_l^i, e_N) and (v_m^j, e_N) to G^B, i.e.,

we add an edge that connect every copy of v_l and and every copy of v_m to e_N. Figure 4.8a gives a 2D geometric constraint graph and figure 4.8b gives its corresponding bipartite graph.

To find the set of irreducible subgraphs, this algorithm uses the following result due to König, and Dulmage and Mendelsohn [21] that gives the single canonical decomposition of any bipartite graph having a perfect matching to a set of irreducible subgraphs:

Theorem 4.5. *(König 1916) Given a bipartite graph $G = (M, N, E)$, every perfect matching of a bipartite graph leads to a unique decomposition into strongly connected components, each one represents an irreducible well-constrained subsystem.*

Proof. [73] □

The computation of a maximum matching in a bipartite graph can be done in polynomial time using any well-known algorithm such as Hopcroft and Karp algorithm [74]. We note that a new method was proposed recently by Barki et al. in [75]: the Re-parameterization. This method can reduce the irreducible at the linear algebra routines level.

The plan of resolution, i.e., the order in which those detected irreducible subgraphs will be solved is obtained as follow:
Let $P \subseteq E$ be the set of edges that are in the resulted matching. A new directed bipartite graph R is constructed as follow:

- Each edge $e \in P$ is replaced by two arcs oriented in both directions.
- Edges $e \notin P$ are oriented from the M to N.
- The strongly connected components are calculated and arcs between them give the partial order of the decomposition.

The decomposition method is presented by the Algorithm 4. Figure 4.8 shows the decomposition of a 2D geometric constraints graph(figure 4.8a) into irreducible subgraphs (figure 4.8b). The plan of resolution is given by figure 4.8c. It means that the subgraph $R1$ then $R2$ and finally $R3$. The corresponding construction steps are as follow:

1. Pin the two points F and G in the plane;
2. Calculate the coordinates of the point E;
3. Calculate the coordinates of the point D;
4. Calculate the coordinates of the points A, B and C by solving the corresponding system of equations.

Algorithm 4 A maximum matching algorithm for GCS decomposition

Input: $G(M, N, E)$: a geometric constraint bipartite graph

Output: R: a decomposition plan

procedure DECOMPOSE(G)

1. Find a perfect matching M of G.

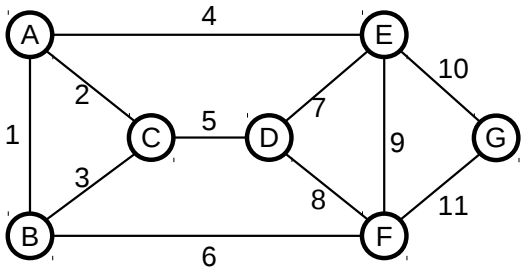
2. Build the directed graph G' from G by replacing each edge (x, y) in M by two arcs xy and yx, and by orienting all other edges from Y to X.

3. Compute the strongly connected components of G'. Each of these strongly connected components is irreducible.

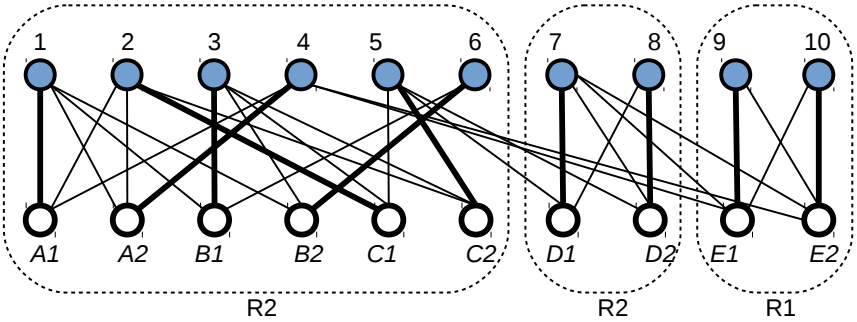
4. To compute the dependency between these irreducible subgraphs, build the reduced graph R from G' by contracting each strongly connected component in a vertex. Each arc of R, say from s_1 to s_2, means: solve subsystem s_2 before $s1$. A compatible total order between subsystems can be obtained by any topological sorting of R.

return R

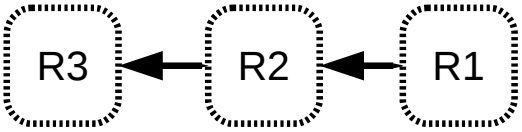
end procedure



(a) A Geometric constraint graph



(b) The corresponding bipartite graph of fig. 4.8a after pinning the edge (F, G), its perfect matching and the set of irreducible subgraphs



(c) subgraph resolution order (DR-plan)

FIGURE 4.8: Maximum matching decomposition algorithm

If the obtained maximum matching is not perfect, then there is an under- or over-constrained parts in the graph. The well-constrained part will be isolated using the following theorem due to Dulmage and Mendelsohn. The over- and under-constrained parts will be returned for further corrections.

Theorem 4.6. *(Dulmage and Mendelsohn, 1958)*

Let $G = (M, N, E)$ any bipartite graph. The set $V = M \cup N$ can be partitioned into three sets D, A, and C defined as follow:

- *D is the set of vertices of G unsaturated by at least one maximum matching;*
- *A is the set of vertices $V - D$, adjacent to at least one vertex of D;*
- *C is the set $V - A - D$.*

These three subsets are unique and lead to a canonic decomposition of G into three subgraphs G_1, G_2, and G_3, defined by:

- *The well-constrained subgraph: $G_1 = (C_1, C_2, E_1)$, where $C_1 = C \cap Y, C_2 = C \cap X$, and E_1 is the set of edges induced by G_1;*
- *The over-constrained subgraph: $G_2 = (D_1, A_2, E_2)$, where $D_1 = D \cap Y, A_2 = A \cap X$, and E_2 is the set of edges induced by G_2;*
- *The under-constrained subgraph: $G_3 = (A_1, D_2, E_3)$, where $A_1 = A \cap Y, D_2 = D \cap X$, and E_3 is the set of edges induced by $G3$.*

Proof. [73] □

4.5.4 Some remarks on the MM Algorithm

In some cases, especially, when the constraint graph is composed of two parts, one of which is over-constrained and the other is under-constrained, algorithm 4 can produce incorrect results. Such a case is illustrated in 2D by the figure 4.9. If the pinned edge is (B, D), the corresponding bipartite graph will have a perfect matching and the constraint graph will be considered well-constrained. (see [76] for more detail). It is very easy to overcome this problem by simply pinning all the edges, one after the other. This modification will increase the time complexity of the algorithm by a factor of m. To prevent this factor, this modification has to reuse the previous calculated maximum matching, every time an edge is pinned.

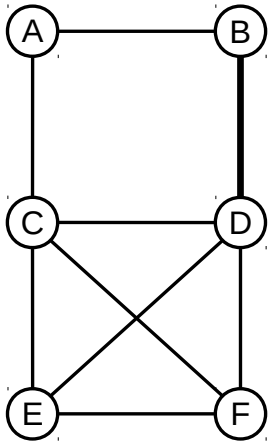
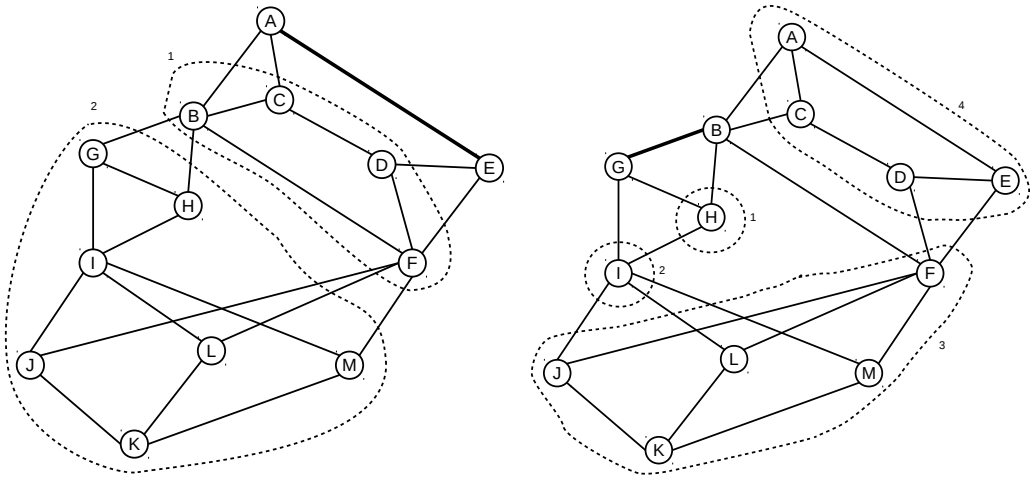


FIGURE 4.9: An example of an incorrect case

The second remark on the MM algorithm concerns the number of detected subgraphs and the size of the largest system of equations to be solved. The example of figure 4.10 shows that it depend on the choice of the pinned edge. In figure 4.10a, where the pinned edge is (A, E), the size of the largest system of equations to be solved is 14 unknown (it correspond to the subgraph :$\{G, H, I, J, K, L, M\}$). But if the pinned edge is (G, B), then, the size of the largest system of equations to be solved is 10 unknown (it correspond to the subgraph :$\{J, K, L, M, F\}$). The geometric elements that correspond to those

subgraphs are represented in figure 4.9 with a dotted line. They correspond to the strongly connected component detected in step 4 of Algorithm 4.They are numbered by the order in which they will be solved. To detect all irreducible subgraphs, i.e. an optimal decomposition, we have to pin all edges, one after another. For each pinning, we have to recalculate the maximum matching, we can avoid this recalculation by reusing data of the previous steps. We note that the resolution order (step 4 algorithm 4) will be no more usable, but we can use for example the Skeletonization algorithme (see section 4.5.6) as a recomposition method.



(a) The pinned edge is (A, E) (b) The pinned edge is (B, G)

FIGURE 4.10: Two different MM-decompositions of the same constraint graph.

4.5.5 Condensing and Frontier algorithms

In this section, we will explain two methods proposed by Hoffman et al. [12] for geometric constraint decomposition. Each one uses the flow-based algorithm discussed in section 4.5.2 as a building block to detect minimal solvable subgraphs. They differ in the process of recomposition.

4.5.5.1 The Condensing Algorithm

Condensed algorithm (CA), proposed by Hoffman et al. [12] is based on the repetitive application of the flow-based algorithms (see section 4.5.2) to the 2D constraint graphs for finding minimal well constrained subgraphs. At each step, the detected subgraph is sequentially extended by adding more geometric objects one at a time, which are rigidly fixed with respect to the subgraph. After a subgraph has been extended, it is then simplified into a single geometric object, and the rest of the constraint graph is searched for another minimal subgraph. Figure 4.11 gives an example of the subgraph extension.

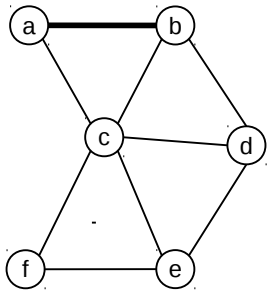


FIGURE 4.11: Constraint graph with vertices of weight 2 and edges of weight 1. The minimal dense subgraph {a, b} can be extended sequentially by the other elements, in alphabetic order.

Consider the constraint graph G of Figure 4.12a. Initially all vertices have weight 2, all edges have weight 1. The vertices connected by the heavy edges constitute minimal or sequentially extended subgraphs. After four simplifications (see figures 4.12b 4.12c 4.12d) the original graph is replaced by one vertex.

4.5.5.2 Frontier Algorithm

A major drawback of the Condensed algorithm is that it can fail in some situations. In some particular cases, the simplification of a minimal or extended

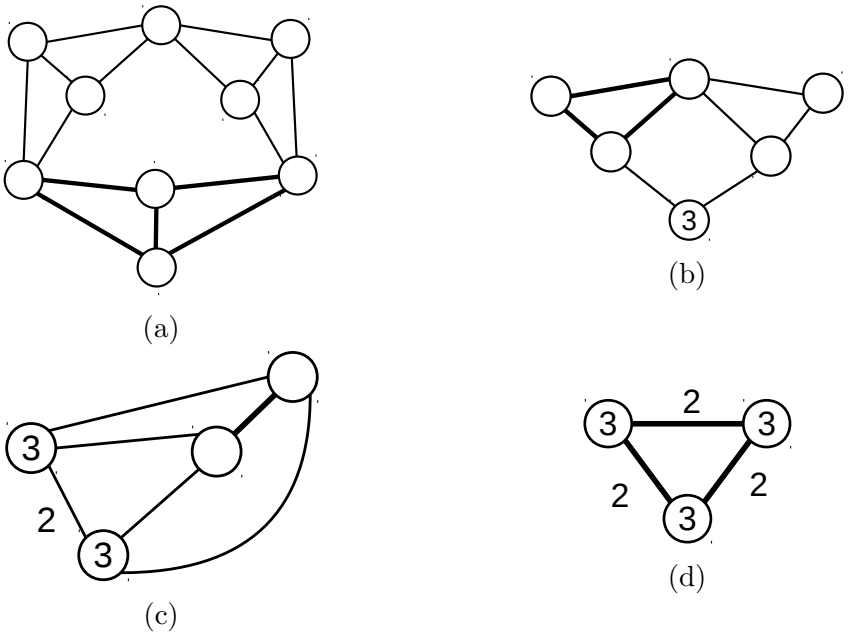


FIGURE 4.12: Condenced Algorithm principle

dense cluster into a single vertex don't preserve the solvability of the graph, because it loses valuable information about the structure of the subgraph. The constraint graph of figure 4.13 represents a $2D$ GCS. The vertex weights are 2; the edge weights are 1. The graphs $ABCDNO$,$EFHGID$ and $NMKLIJ$ are all solvable. Suppose that first subgraph $S_1 = ABCDNO$ was found and simplified into one vertex S of weight 3. Now the graph $SEFHGI$ is not solvable anymore.

Frontier algorithm can be considered as an enhancement of the Condensed algorithm. The way to find minimal subgraphs and their sequential extensions is similar to that of Condensed algorithm. The only difference between the two algorithms is the simplification step. In the Frontiers algorithms, only the vertices that are not connected to outside the subgraph are simplified to a single vertex c_i . The weight of c_i is D (D = 3 in 2D space). The vertex c_i is connected to each frontier vertex v by an edge whose weight is equal to the weight of v. In another version of Frontier algorithm, called MFA (Modified Frontier algorithm), the vertex c_i is connected to each frontier vertex v by a

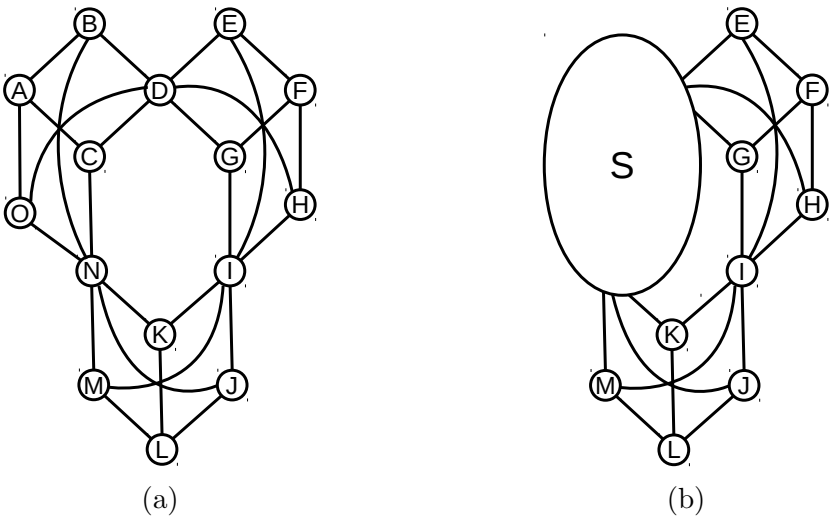


FIGURE 4.13: An example where the CA simplification step fail.

combined edge whose weight is the sum of the weights of the original edges connecting internal vertices to v. The frontier vertices, the edges connecting them, and their weights remain unchanged. Figure 4.14 gives an example of graph reduction using the Modified Frontier algorithm.

4.5.6 Recursive Skeletonization Algorithm

In their paper [9], Ait-Aoudia and Foufou presented an $O(n^3(n+m))$ decomposition/recombination algorithm, called S-DR. Comparing to CA or MFA, S-DR planner proceeds in an easiest way in the process of simplifications and abstractions. The algorithm uses a graph reduction method to solve systems of 2D geometric constraints, based on a key concept called skeleton. S-DR planner figures out a plan for decomposing a well-constrained system into small sub-systems and recombines the solutions of these sub-systems to derive the solution of the entire system. This algorithm uses the same method (see section 4.5.2) to find solvable subgraph as in Condensed or Frontier algorithm. S-DR planner will find a sequence of graphs G_i $(G_1, G_2, ..., G_k)$. G_1 is the initial constraint graph G. Every G_i containing several subgraphs (called

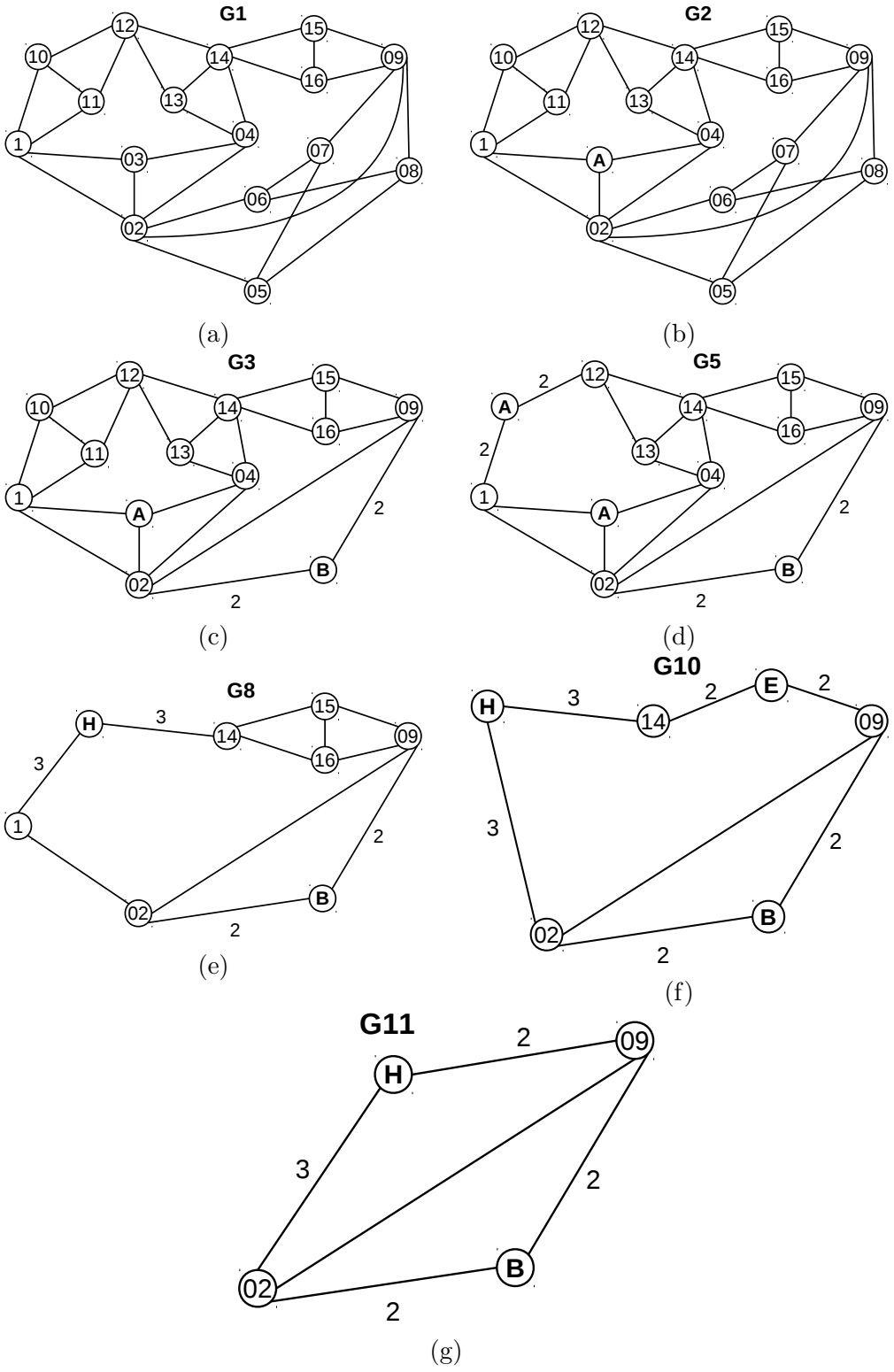


FIGURE 4.14: Example of Frontier algorithm

clusters) is simplified into a graph $G_{i+1} = T_i(G_i)$ where T i is the transformation (simplifier map) giving the skeleton of G_i (G_{i+1} is the skeleton of G_i). If the constraint graph G is structurally well constrained, then the process terminates when the graph G_k consists of one basic cluster. The main procedure is given by Algorithm 6, the skeleton procedure is given by Algorithm 5. Figure 4.15a gives an example of a constraint graph and its clustering; figure 4.15b shows the corresponding skeleton and figure 4.16 illustrate the clusters placement process.

Algorithm 5 The recursive Skeletonization.

Input: G: a geometric constraints graph; $S = \{C_1, C_2, ..., C_n\}$: the set of detected cluster ;

Output: The skeleton graph $G_s(E_s, V_s)$.

```
procedure SKELETONIZE(G, S)
    V_s ← ∅
    for all C_i ∈ S, C_j ∈ S, i ≠ j do
        V_s ← V_s ∪ (C_i ∩ C_j);
    end for
    for each C_i ∈ S do
        X ← C_i ∩ V_s;
        let X = {n_1, n_2, ..., n_j}, triangulate X by doing:
            E_s ← E_s ∪ {(n_1, n_2)}
            for k ← 3, j do
                E_s ← E_s ∪ {(n_k, n_{k-1}), (n_k, nk − 2)}
            end for
    end for
    return G_s
end procedure
```

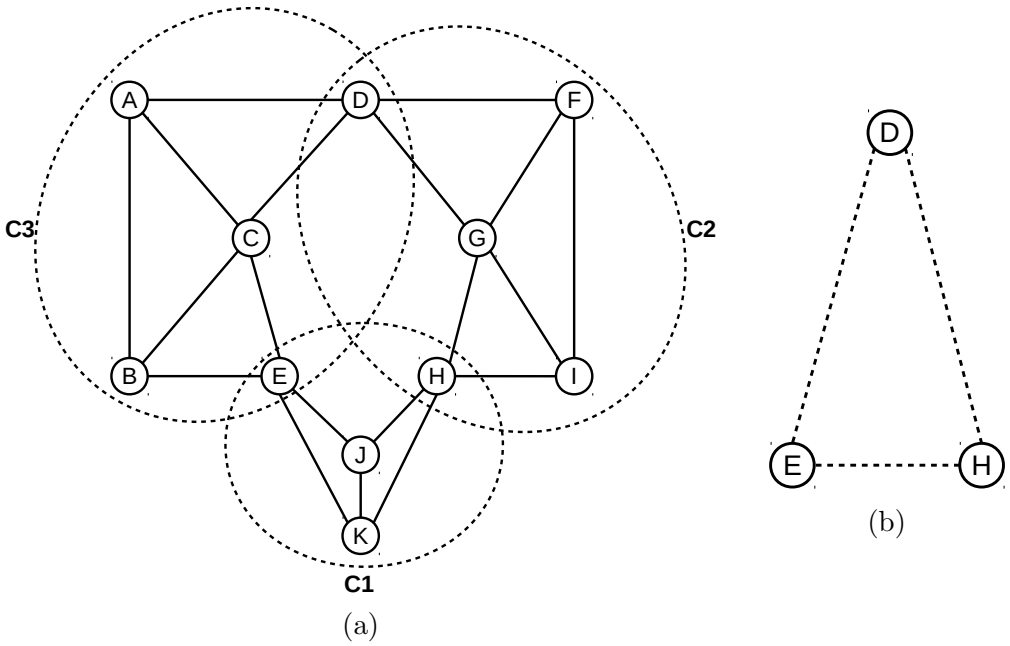


FIGURE 4.15: Example of constraint graph clustering and its skeleton

Algorithm 6 Geometric constraint graph decomposition

Input: G: a geometric constraints graph;

$i = 1; G_1 = G;$

procedure SOLVE(G_i)

$S \leftarrow CLUSTERS(G_i)$;

/*Clusters are calculated using the flow-based method, section 4.5.2*/

if $|S|>1$ **then**

$G_{i+1} \leftarrow skeletonize(G_i, S)$;

Solve(G_{i+1})

Place clusters of G_i relatively to G_{i+1}

else

Place the nodes of G_i in the plane;

end if

end procedure

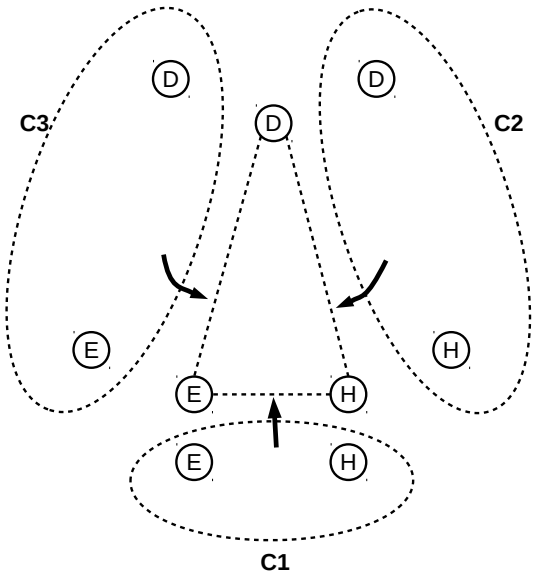


FIGURE 4.16: The clusters placement for the skeleton of figure 4.15a.

4.6 Conclusion

In this chapter, we have studied two main categories of solvers, also called planners, in the sense where they give a plan of resolution represented by a set of minimal subgraphs and their order of geometric construction.

The first category is the constructive solvers, called also shape recognition solvers. In this class, the only basic subgraphs are always triangles. Those algorithms use the inherited methods from graph theory to detect triangles in the corresponding geometric constraint graph. Despite the constructive methods cover a large sub-class of the ruler and compass constructible GCS, which are the most encountered the CAD world, they have a severe limitation: it is very difficult to generalize these methods. Irreducible subgraphs should always be triangles, and their extension requires a specific algorithm for each new type of shape other than a triangle.

The second category is the general solvers, where the minimal subgraph can be any irreducible well-constrained graph. Those methods use a flow-based algorithm to find minimal solvable subgraphs. By some structural analysis,

those methods can calculate the plan of resolution. Another type of general solvers, based on the maximum matching calculation, was also presented. Those algorithms have the advantage of solving any GCS, but major of them are discussed or compared theoretically, without giving any tests highlighting their performances. In the next section, we present a design of a simple, but efficient random generator of 2D geometric constraint graph. It can be used to make benchmarks for consistent tests, or observe the behaviour of these two categories of solvers.

Chapter 5

Our Contribution: A 2D Geometric Constraint Graph Generator

Contents

5.1 Introduction 75

5.2 Geometric Constraint Graph Decomposition. . . 77

5.3 Generating well-constrained graphs 78

5.4 Random constraint graph generator 86

5.5 Embedding over- and under-constrained subgraph 95

5.6 Experimental results 95

5.7 Conclusion 97

5.1 Introduction

Tests and analysis of graph-based solvers are usually done in a theoretic manner or with well-chosen examples. However, to make consistent tests, or observe the algorithm behaviour on graphs with various sizes and structural properties, that can affect the behaviour of the solver, there is a great need for a tool that can produce such graphs. It can help performing tests with more data analysis rather than pure theory. This need is due to several factors, mainly, it is very hard to manually create non-decomposable graphs. Until now, there is no known method for creating this category of graphs.

Our contribution is the design of a simple, but efficient random geometric constraint graph generator. This is significant because to the best of our knowledge, there is no such generator so far. It should be of interest in the areas of 2D geometric constraints solving. Our generator can be used to make benchmarks for consistent tests, or observe the behaviour of 2D geometric constraints solving algorithms. It can produce problem instances with various sizes and structural properties, covering different cases of complexity.

We restrict ourselves to 2D geometric constraint systems in the generic sense; we focus only on the solvability of the graph and ignore the numerical values of the geometric constraint. We focus on creating graphs for 2D well-constrained geometric problems for which the Laman condition [18] holds. Two algorithms will be sketched. The first one is dedicated to the creation of non-decomposable graphs, or graphs with a given property, called the degree of decomposability, that we have introduced. The second algorithm, which represents our proposed generator, allows the creation of a graph with the following desired properties:

Completeness it can generate all possible 2D geometric constraints graphs, i.e., any graph satisfying the Laman condition [18] can be generated. This property is very important because it characterizes the domain of geometric constraint systems covered by our proposed generator.

Customizability: it generates some particular configuration, targeting special situation of difficulties. This is done by parametrizing the generator. It can be parametrized to produce a graph solvable by SR-Planners or only by MM-Planners. Moreover, it can generate a particular domain of GCS (representing a sub-class of ruler and compass constructible GCS) or a general one. By specifying what we called a degree of decomposition, we can build a graph that has a low or a high number of solvable subgraphs. We can also determine the average size of the smallest subgraph. Those two parameters represent our adopted metric of solvability. They characterise the computational time complexity of a GCS.

Simplicity and efficiency: our generator is easily understood; it is straightforward implemented based on a verified theory.

We start by presenting some notions about the geometric constraint graph, its decomposition, and its generic solvability. Then we outline the generation of random well-constrained graphs and show some tests on their decomposability. Then, we introduce a suitable graph generator for well-constrained problems. In the end, we present some experiments to evaluate the solvability of generated graphs.

5.2 Geometric Constraint Graph Decomposition.

Decomposition process can be done in two steps; the first one can be considered as a pre-processing that consists of isolating the unsolvable parts: the over and under-constrained subgraphs. For this purpose, we can use for example the algorithm proposed in [10]. The obtained well-constrained graphs, which are the solvable ones, are then decomposed in the second step. The goal of this decomposition is to speed up the resolution process, by providing a plan of resolution. We focus only on the decomposition of the well-constrained parts. Transforming the whole geometric constraint problem into a system of equations, and solving it, is extremely time-consuming. The main goal of graph-based techniques is to decompose the problem into small sub-systems. Each one is solved separately with respect to an order given as output by the DR-planer. Two parameters that we call decomposability metrics can decide how fast will be the resolution process:

1. The size of the largest subgraph as proposed by Hoffman et al. [15].
2. The number of detected subgraphs over the total number of vertices [1]. We can formulate this by a parameter, d, called the degree of decomposability, where $d = n/g$. n is the number of vertices of the geometric constraint graph and g is the number of detected subgraphs to be solved directly by algebraic methods. d indicates if the graph is highly or weakly decomposable.

In the rest of this section, we show how to design a random graph generator that uses those two parameters as input. Our approach is organized in two steps: in the first one, we present a procedure called RH that generates a non-decomposable graph, i.e., a graph that has a known size and no detectable

subgraphs. Then we use the procedure RH to develop our random graph generator, called RRH, which can produce a graph with a known size of the largest subgraph and a known number of subgraphs. RRH can also create graphs that are solvable by SR-Planner or by only MM-Planners.

5.3 Generating well-constrained graphs

In [77], Tay and Whiteley presented an inductive construction that always produces a well-constrained graph. This construction is due to Henneberg [19]. The definition follows.

Definition 5.1. Starting with a graph $G = K_3$, a well-constrained graph can be built inductively by adding one vertex at a time using one of these two Henneberg operations:

1. Operation HI : add a new vertex v to G, then connect v to two chosen vertices u and w from G via two new edges (v, u) and (v, w). See Figure 5.1.

2. Operation HII : add a new vertex v to G, chose an edge (y, w) and a vertex u from G, then add three edges (v, u), (v, w) and (v, y) to G, finally delete the edge (y, w). See Figure 5.2.

The following definition justifies our interest in Henneberg construction.

Definition 5.2. [77] A graph G is well-constrained if and only if it has a Henneberg construction.

Proof. ([78]) □

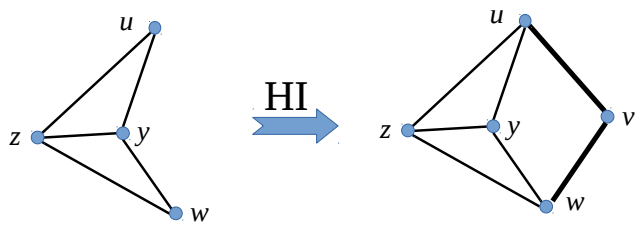


FIGURE 5.1: The first operations of Henneberg construction: HI

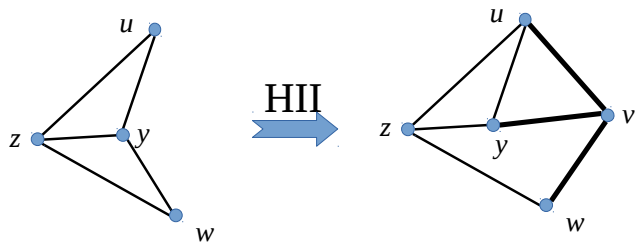


FIGURE 5.2: The second operations of Henneberg construction: HII

Henneberg construction always generates a well-constrained geometric constraint graph. In reviewing the literature, no study was done on the relation between the choice of Henneberg steps, HI and HII, and the structural property of the generated graph. How many solvable subgraphs has the generated graph if we always chose the first operation HI or the second operation HII, or if the two operations have the same or different probabilities of being selected? Let p be the probability of choosing HI. Algorithm 7 allows the generation of well-constrained graphs for different values of p.

Note: in step 3 of the procedure RH, we use an uniform pseudorandom number generator presented by Matsumoto and Nishimura [79].

To evaluate the decomposability degree of the graph, generated by the procedure RH of algorithm 7, for different values of p, three sets of 2000 random graph were generated. Each set corresponds respectively to the value: 0, 0.5 and 1 of the parameter p. Due to the randomness, 20 instances for each size ranging from 4 to 100 vertices were generated, and the mean has been calculated.

Algorithm 7 Generate a random well-constrained graph using Henneberg construction according to the probability p.

Input: n: the size of the graph; p: the probability of choosing HI;

Output: a well-constrained graph G.

```
1:  procedure RH(n, p)
2:      start with an initial graph G = K_3
3:      for i ← 1, n − 3 do
4:          Generate a random number r, uniform on [0, 1);
5:          if r < p then
6:              add a new vertex v to G according HI;
7:          else
8:              add a new vertex v to G according to HII;
9:          end if
10:     end for
11:     return G
12: end procedure
```

We have calculated the degree of decomposability using an MM-Planner, based on the Dulmage-Mendelsohn decomposition, presented by Ait-Aoudia et al. [10]. This planner has been chosen because it always returns a decomposition if one exists [8]. The result of this decomposition depends on the randomly chosen edge that was fixed (see section 4.5.4).

As it can be seen in Figure 5.4, for $p = 0$, which means that we regularly use the second operation of Henneberg construction: HII, generally, graphs for size >40 were not decomposable. For example, the graph of size 20, only 30% were not decomposable, but for the size 40 this percentage increases approximately to 80%. In our tests, for the 20 randomly generated graphs, with a size ranging from 40 to 100, rarely there existed a decomposable one. This is due to the use of the second operation of Henneberg, which splits edges. This operation may merge two subgraphs in a single non-decomposable one

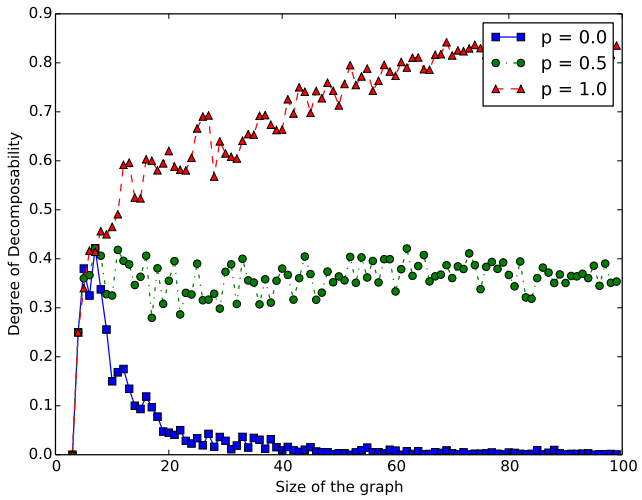


FIGURE 5.3: The size of the graph vs degree of decomposability for p = 0, p = 0.5 and p = 1.

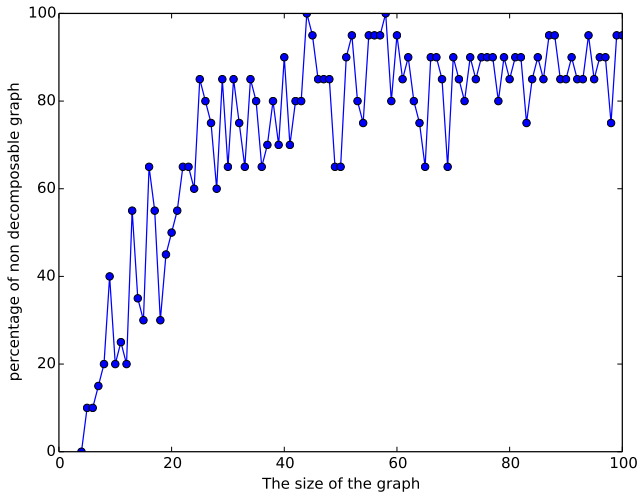


FIGURE 5.4: The percentage of non decomposable graphs in each set of 20 instances generated, size ranging from 4 to 100, the probability $p = 0$

at each step. This result can be used in the generation of non-decomposable graphs of more than 40 vertices. Figure 5.3 plots the size of the graph vs degree of decomposability, for p = 0, p = 0.5 and p = 1. Three graphs of size 50 generated by RH, for the case of $p = 0.0$, $p = 0.5$, $p = 1.0$ is given by Figure 5.6. In the case of $p = 1$, where only the first step of Henneberg construction was used, whatever the size of the generated graph, it was decomposable. For $p = 0.5$, the degree of decomposability was approximately 0.4. Table 5.2 gives the different values of d, for the graph size, ranging from 100 to

Size	$p = 0.0$	$p = 0.5$	$p = 1.0$
05	0.338	0.406	0.456
10	0.135	0.388	0.596
15	0.078	0.381	0.581
20	0.028	0.330	0.580
25	0.016	0.329	0.568
30	0.036	0.400	0.641
35	0.032	0.311	0.674
40	0.006	0.360	0.750
45	0.005	0.374	0.759
50	0.005	0.351	0.755
55	0.003	0.399	0.796
60	0.002	0.365	0.810
65	0.004	0.368	0.818
70	0.001	0.411	0.829
75	0.002	0.379	0.847
80	0.001	0.321	0.763
85	0.010	0.351	0.840
90	0.002	0.369	0.824
95	0.002	0.351	0.817
100	0.002	0.335	0.811

TABLE 5.1: Decomposability degree for different values of p : 0, 0.5 and 1.

1000 vertices. We have randomly generated five instances for each size and calculated the mean. Results show that the degree of decomposition depends on the probability value p rather than on size. In Figure 5.5, we have plotted the decomposability degrees for p ranging from 0 to 1. It can be seen that the degree of decomposability increases in the direct proportion of p, i.e., the more we decrease the probability value p, the more we weaken the values of the decomposability of the graph. To generate a non-decomposable graph, p must be equal to 0. Because of algorithm 7 employs a degree of randomness as part of its logic, in many cases, for $p = 0$, the generated graph may have more then one subgraph.To ensure that the generated graph has no more than itself as a non-decomposable subgraph, we can use Algorithm 8 instead. To calculate the number of irreducible sub-graphs (step 5 of algorithm 8), We have used the MM-Planner [10] presented in section 4.5.4. We not that in our

Algorithm 8 Generate a non decomposable well-constrained graph.

Input: n: the size of the graph;

Output: a non decomposable well-constrained graph G(irreducible).

```
procedure RH0(n)
    start with an initial graph G = K_3
    repeat
        Generate a graph G using RH(0,n);
        s ← the number of irreducible subgraphs of G
    until s = 1                    ▷ Only G itself as irreducible subgraph
    return G
end procedure
```

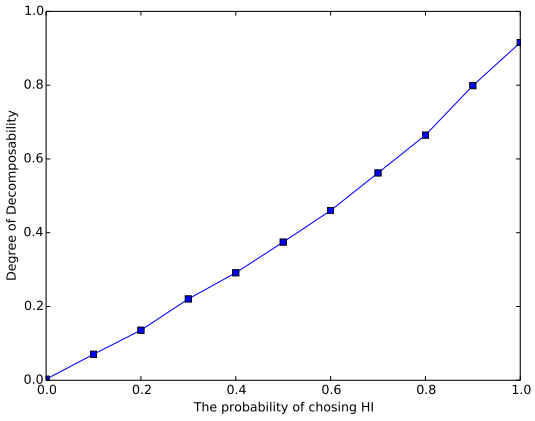


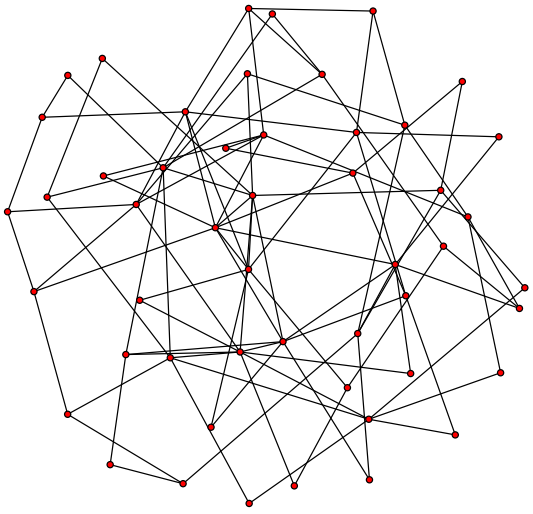
FIGURE 5.5: The probability of choosing the first operation of Henneberg vs the degree of decomposition.

experiment, Algorithm 8 succeed in a bounded amount of time. To calculate the number of irreducible subgraphs, we can use the Algorithm 4 and take into account the remark about the pinning of all edges as suggested in section 4.5.4 .

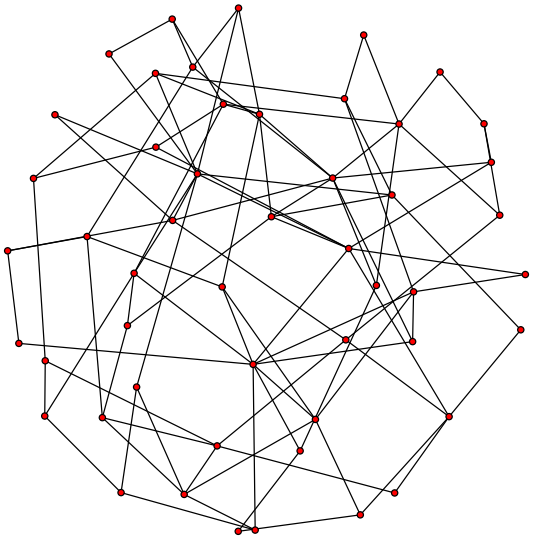
Tests conclude that the parameter p can be used to parameterize the generator to produce highly or weakly decomposable graphs. In the next section, procedure RH of algorithm 7 will be used to generate non-decomposable graphs.

n	p										
	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
100	0.010	0.062	0.122	0.206	0.276	0.366	0.382	0.576	0.654	0.752	0.862
200	0.005	0.063	0.134	0.184	0.294	0.400	0.456	0.511	0.659	0.787	0.853
300	0.003	0.073	0.121	0.249	0.299	0.351	0.468	0.553	0.617	0.785	0.917
400	0.003	0.074	0.133	0.214	0.299	0.354	0.488	0.547	0.712	0.814	0.911
500	0.002	0.072	0.144	0.234	0.295	0.409	0.483	0.579	0.657	0.825	0.930
600	0.002	0.068	0.145	0.241	0.290	0.367	0.481	0.568	0.641	0.800	0.928
700	0.001	0.072	0.133	0.223	0.309	0.386	0.467	0.584	0.665	0.811	0.939
800	0.001	0.078	0.148	0.231	0.277	0.372	0.475	0.559	0.702	0.793	0.954
900	0.001	0.076	0.144	0.212	0.288	0.359	0.447	0.551	0.653	0.810	0.946
1000	0.001	0.067	0.134	0.211	0.287	0.382	0.454	0.593	0.686	0.811	0.915

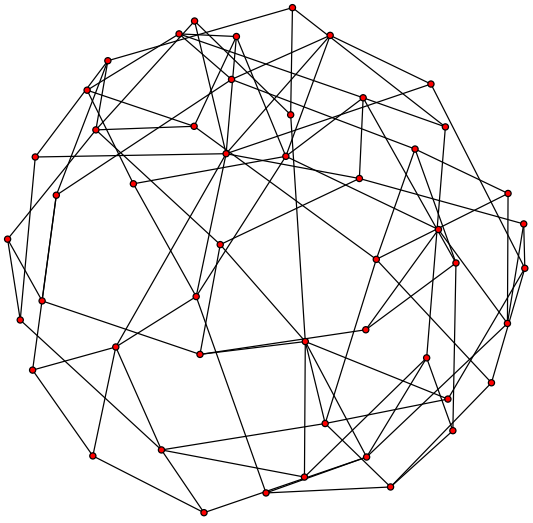
TABLE 5.2: The degree of decomposition for different values of p and size of the graph



(a) $p = 1$.



(b) $p = 0.5$



(c) $p = 0$

FIGURE 5.6: Three Geometric constraint graphs generated by *RH* for different values of p

Case	$p = 1.0$	$p = 0.5$	$p = 0.0$
1	40	14	1
2	37	19	1
3	39	13	1
4	36	20	1
5	40	27	1
6	37	16	2
7	38	20	1
8	37	24	1
9	39	13	1
10	38	18	1

TABLE 5.3: Number of detected subgraphs with $n = 40$ and for different values of p : 0, 0.5 and 1.

5.4 Random constraint graph generator

The central role of a planner is the decomposition of geometric constraint graphs. To evaluate how planners deal, or how will be their behavior with some types of graphs, a tool for generating situation with some desirable properties can be very helpful. Such tool enables the evaluation of any solving methods with more data analysis rather than pure theory. First, we show how to generate graphs that are decomposable by SR-Planners, and then we explain how to set the size of the largest subgraph to a desirable value. Decomposability degree is always represented by the probability value p. If we want to limit the graph generated by the procedure RH, presented in the previous section, to only those that are decomposable by SR-Planners, where the smallest subgraphs are limited to triangles, the first idea would be to use only the first operation HI. We will prove that is incorrect, and then we propose the procedure RRH (Algorithm 9), which is a recursive replacement of edges by graphs that are generated by the procedure RH of the Algorithm 7. The goal of this modification is to generate the class of graphs solvable by the SR-Planners using only HI.

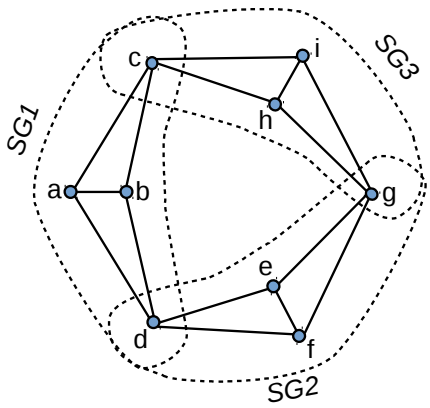


FIGURE 5.7: A geometric constraints graph.

Figure 5.7 shows a graph that can be decomposed by any SR-Planner. Let us show if there exist a Henneberg construction for this graph that uses only the first operation HI. Suppose that we start with the triangle a, b, c as the first step, the second step must be no other than adding the vertex c. After this stage, no other vertex can be added to the cluster. If we start with another triangle than a, b, c, we will obtain the same reasoning because the graph is formed by three symmetric parts: SG_1, SG_2 and SG_3 . Hence, this graph is solvable by SR-Planners and not constructible by procedure RH using only the first operation HI. Next, we show how to add a recursion mechanism to the procedure RH so that any graph decomposable by SR-Planners can be generated using exclusively HI, i.e., $p = 1$. This addition will simplify the design of our generator. We have already seen, that for $p = 0$, graphs of size > 40 generated by procedure RH are mostly non-decomposable. Hence, we can produce a larger graph by the composition of non-decomposable ones. This method can be used to create a graph that has a predefined size of the largest subgraph as required by the metric defined in section 5.2. Generating a graph G is a sequence of graphs $G_1, \cdots, G_n$ with the following properties:

1. $G_1 = RH(m, p)$,

2. $G_n = G$

3. G_{i+1} is obtained from G_i, through the replacement of randomly chosen edge of G_i, by a new subgraph $RH(m, p)$.

We have to prove that the replacement of a randomly chosen edge of G_i, by a new subgraph $RH(m, p)$ always leads to a well-constrained graph. We note that the starting graph for the Henneberg construction is K_3. It can be easily adapted to start with a graph with only one edge (K_2); we just need to use HI in the first time of the Henneberg sequence.

Before that, we need the following lemma due to Haas et al. [78]..

Lemma 5.3. *[78] A Laman graph has a Henneberg construction starting from any prescribed subset of two vertices.*

Proof. See [78] for a detailed proof. □

Theorem 5.4. *Let $G = (V, E)$ and $H(V_H, E_H)$ be two Laman graphs. Let $(a, b) \in E$ be any edge of G, and $a' \in V_H$ and $b' \in V_H$, two vertices of H. The graph $G^* = (V^*, E^*)$, that result from the replacement of the edge (a, b) by the graph H obtained using the following operation is also Laman:*

- $V^* = V \cup V_H \setminus \{a, b\}$
- $E^* = E \cup E_H \cup S$, *where* $S = \{(a', x) : (a, x) \in E\} \cup \{(b', x) : (b, x) \in E\} \setminus (\{(a, b)\} \cup \{(a, x) : (a, x) \in E\} \cup \{(b, x) : (b, x) \in E\}\})$. *i.e., for every neighbour x of the vertex a (resp. b) we add an edge (a', x) (resp. (b', x)) to G^*, then we delete the two vertices a and b.*

Proof. By definition 5.2, in order to prove that G^* is a Laman graph, we have to prove that it has a Henneberg construction. Because G is supposed to be Laman graph, then, it has a Henneberg construction, let be : G_{steps}. Because H is supposed to be Laman graph, and by lemma 5.3, it has a Henneberg

construction: H_{steps}, that starts from the two nodes a' and b'. We conclude that the Henneberg construction of the graph G^* will be G_{steps}, followed by H_{steps} starting from the edge (a, b). Hence, G^* is a Laman graph. □

If m is the desired size of the largest subgraph, then, Algorithm 9 can be used to generate a random well-constrained graph, according to the two metrics: (1) the size of the largest sub-graph, (2) the degree of decomposition. Figure 5.8 shows an example of the different steps to generate a graph solvable only by MM-Planners.

Notice, as seen in section 5.3, generally the procedure RH generates a non-decomposable graph for $m > 40$ and $p = 0$. For $m < 40$, mostly, procedure RRH will be not efficient. In this case, instead of generating non-decomposable graphs using $RH(m, p)$, we can easily design a dataset of non-decomposable graphs, having a size lower than 40 [80]. Step 4 of the procedure RRH will be replaced by a random retrieval from this dataset. If we set the parameter m to a desirable value of the size of the largest subgraph, the parameter p must be equal to zero.

Algorithm 9 Generation of a random well-constrained graph, according to the two metrics: (1) the size of the largest subgraph, (2) the degree of decomposition.

Input: n: the size of the graph, p: the probability of choosing the first step of Henneberg construction and m: the size of the largest subgraph;

Output: $G(V, E)$: a well-constrained graph.

1: **procedure** $RRH(n, p, m)$
2: start with an initial graph $G = K_3$
3: $k \leftarrow n/m$
4: **for** $i \leftarrow 1, k$ **do**
5: Generate a random graph $H(V_H, E_H)$ of size m using $RH(m, p)$;
6: Pick an edge (x, y) from G;
7: Replace the edge (x, y) by the graph H as follows:
8: pick two random vertices x' and y' from the graph H;
9: connect x', to all neighbours of x;
10: connect y', to all neighbours of y;
11: delete x and y;
12: $V \leftarrow V \cup V_H$;
13: $E \leftarrow E \cup E_H$;
14: **end for**
15: Complete G by adding a subgraph $H = RH(n - |V| + 2, p)$ as in steps 5-11.
16: **return** G
17: **end procedure**

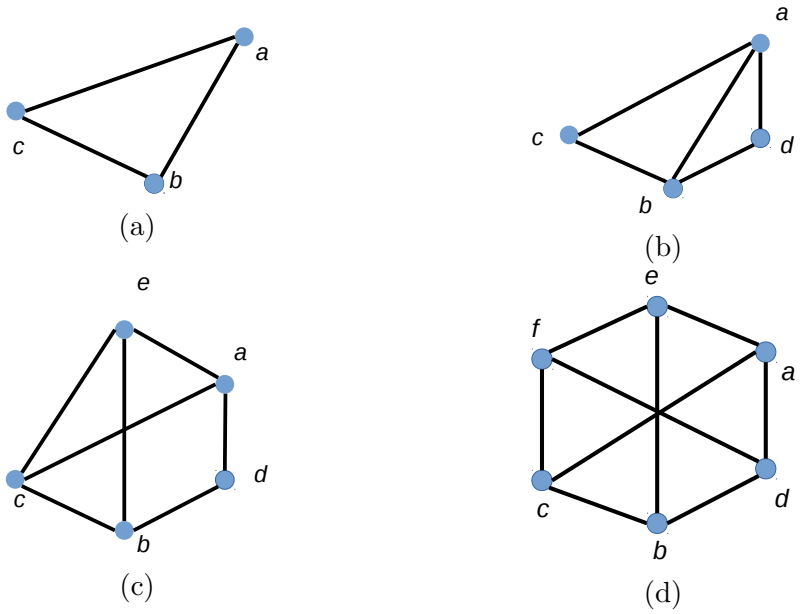


FIGURE 5.8: Four steps to generate a well-constrained graph solvable only by MM-Planners.

(a) the initial triangle. (b) adding a vertex d using the first operation of Henneberg HI. (c) adding a third vertex e using the second operation of Henneberg HII, by deleting the edge (a, b) and connecting e to a,b and c. Adding the vertex f using HII, by deleting the edge (e, c) and connecting f to e, c and d.

Claim 1. Any structurally well-constrained graph can be generated by RRH.

Proof. Since RRH uses Henneberg construction, and any one of the two operations (HI or HII) may be executed in each step of RRH, and by the definition 5.2, we can conclude that RRH generates all the domain of the well-constrained graph in 2D. □

Claim 2. A geometric constraints graph is solvable by any SR-Planner if and only if $p = 1$.

Proof. $\Leftarrow$ If $p = 1$, then the procedure RRH will start initially by a triangle, and recursively replace an edge by a new graph H, generated by the procedure RH. Because $p = 1$, RRH uses only the first operation of Henneberg: $H1$. We recall that all SR-planners solve only a sub-domain

of GCS. In [81], Joan-Arinyo et al. studied the domain of SR-planners, they have proved in that Owen's SR-planner proposed in [14] and the planner proposed by Fudos and Hoffmann in [11] solve the same domain of GCS. Recently, he proposed in [16] a new SR-Planner that also solves the same domain. In order to prove the claim 2, we have to prove that if a graph G is solvable by an SR-Planner (called reduction analysis) proposed by Fudos and Hoffmann in [11], then, it can be generated by procedure RRH for $p = 1$. Let us recall now the principle of the reduction analysis algorithm. More details can be found in [11]. Given the constraint graph $G = (V, E)$, we consider the initial set of clusters $S_G^0 = \{\{u, v\}/(u, v) \in E\}$. Cluster sets, that have a central role to this method, are rewritten using a reduction $\rightarrow$. The reduction $\rightarrow$ is formally defined as follows: Let S_G^k be a set of clusters in which there are three clusters C_1, C_2, and C_3 such that: $\exists g_1, g_2, g_3 \in V, C_1 \cap C_2 = \{g_1\}; C_2 \cap C_3 = \{g_3\}; C_1 \cap C_3 = \{g_3\}$ and $g_1 \neq g_2 \neq g_3$. (i.e., those three clusters pairwise intersect in a singleton). Then: $S_G^k \rightarrow S_G^{k+1}$, where $S_G^{k+1} = S_G^k - \{C_1, C_2, C_3\} \cup \{C_1 \cup C_2 \cup C_3\}$.

The solving process is a repetitive application of reductions $\rightarrow$, that starts from the initial cluster S_G^0. If this process end with a singleton $\{V\}$, i.e., the final cluster that contains only one element, which is the set of nodes V, then the graph G is solvable by the reduction analysis algorithm.

Because $p = 1$, the procedure RRH always uses the first operation HI (in step 5 of Algorithm 9). The generation process of the graph G by RRH is a sequence of graphs: $G_1, ..., G_n$, with the following properties : $G_1 = K_3$; $G_n = G$; and G_{i+1} is obtained by replacing an edge $e = (a, b)$ of G_i by a graph $H(V_H, E_H)$, which is generated using the first procedure RH. We proof by induction that if there exist a sequence of reductions for G_i, then there also exist a sequence of reductions for G_{i+1}.

(a) G_1 is a triangle; hence it is decomposable by the reduction analysis algorithm (trivial).

(b) Suppose that there exist a reduction $\rightarrow$ for $G_i(V_{Gi}, E_{Gi})$. G_{i+1} is obtained by replacing an edge $e = (a, b)$ of G_i by a graph $H(V_H, E_H)$, which is generated using the first procedure RH. We show how the reduction of the graph G_{i+1} by $\rightarrow$ is done, by first reducing its subgraph H, and then, reducing G_i which is supposed reducible. We first prove that H is reducible by $\rightarrow$:

H is created by the procedure RH as follow: we start by a triangle H_0, H_{i+1} is obtained by adding a new vertex v to H_i and connecting it to two randomly chosen vertices of H_i: u and w ($p = 1$, we use only HI). Suppose that H_i is reducible by $\rightarrow$, then it corresponding set of clusters can be reduced to set S_{H_i} of a single cluster C_{Hi} containing u and w. $C_{Hi} \cap \{u, v\} = \{u\}$. $C_{Hi} \cap \{w, v\} = w$ and $\{u, v\} \cap \{w, v\} = v$. Hence, the set of clusters: $\{C_{Hi}, \{u, v\}, \{w, v\}\} \rightarrow \{C_{Hi} \cup \{u, v\} \cup \{w, v\}\} = S_{H_i+1}$. We deduce that the graph H is reducible by $\rightarrow$. The final result of the reduction process (as presented by Fudos and Hoffmann) is the single set composed of the vertices of H, let be $S_H = \{V_H\}$.

To reduce G_{i+1}, initially, we form a set S^0_{Gi+1} from G_{i+1}. For each edge $e = (u, v)$ in G_{i+1}, there is a cluster $C = \{u, v\} \in S^0_{Gi+1}$. The reduction steps S^0_{Gi+1} will be as follow:

First, reduce the clusters of S^0_{Gi+1} that correspond to the edges of H ($E_H \subset E_{Gi+1}$). They will be reduced to a single set $= V_H$. We obtain a set of clusters S^k_{Gi+1}.

G_i is supposed decomposable by the reduction analysis, let S^0_{Gi} be its initial set of clusters.

Because G_{i+1} is obtained by replacing an edge (a, b) of $G_i(V_{Gi}, E_{G_i})$ by the graph $H(V_H, E_H)$, then, $V_{G_i} \cap V_H = \{a, b\}$. Because $(a, b) \in$

E_{Gi}, then, $\{a, b\} \in S^0_{Gi}$. If we replace the cluster $\{a, b\}$ in S^0_{Gi} by V_H, we obtain S^k_{Gi+1}. We know that $a \in V_H$ and $b \in V_H$. Hence, any merge of the cluster $\{a, b\} \in S^0_{Gi}$ will correspond to a merge of the cluster $V_H \in S^0_{Gi+1}$. Hence the reduction of S^k_{Gi+1} can be done with the same sequence of reductions of S^0_{Gi}, and end with a single cluster. We deduce that G_{i+1} is decomposable by the reduction analysis. Hence, it is decomposable by any SR-Planner.

Finally, we conclude that if $p = 1$, then a graph G generated by the procedure RRH is solvable by any SR-Planner.

$\Rightarrow$ Now we prove that if a graph G is solvable by any SR-Planner, then G can be generated by the procedure RRH with $p = 1$. Let G be a graph that is decomposable by the reduction analysis algorithm. Then, $S_{G_1} \rightarrow SG_2 \rightarrow ...S_{G_{k-1}} \rightarrow S_{G_k}$ is the corresponding cluster reduction. We prove that there is a sequence of steps in the procedure RRH that generates G. S_{G_K-1} is the penultimate set of clusters of the reduction analysis, then it has three elements : L, M, N, where $L \cap M = g_1$; $L \cap N = g_2$;$M \cap N = g_3$. The last step of the reduction corresponds to the first step of the procedure RRH. (step 1 in algorithm 9. It creates a triangle, let be $\{g_1, g_2, g_3\}$. The three edges of this triangle: $(g_1, g_2), (g_1, g_3), (g_2, g_3)$, will be replaced respectively (steps 4 and 5 of algorithm 9) by three graphs: G_L, G_M and G_N that corresponds to the three clusters: L, M and N. If a cluster C has only two elements, i.e., it corresponds to an edge, this edge will not be replaced by any graph, but considered as a graph composed by only one edge. Because the three graphs G_L, G_M and G_N are well-constrained, every one of them has a reductions sequence, and the last reduction of each sequence has a corresponding step in the procedure RRH that can be demonstrated in the same manner as the last reduction sequence of the graph G shown above. Because there is a finite number of reductions, we deduce that, if

n	p									
	50	100	150	200	250	300	350	400	450	500
500	5.850	3.750	3.000	2.850	2.300	2.050	1.450	1.400	1.700	1.100
600	7.300	3.550	2.450	2.950	2.600	2.050	2.400	2.100	1.050	1.200
700	5.800	3.350	3.250	2.950	2.800	2.450	2.150	2.000	2.000	2.000
800	7.800	4.500	3.400	2.900	3.100	2.600	2.600	2.050	2.250	2.050
900	7.500	3.500	3.600	3.000	2.600	2.800	2.650	2.650	2.000	2.000
1000	8.600	3.550	3.650	3.250	2.700	2.900	2.600	2.450	2.700	2.200

TABLE 5.4: The average number of subgraph detected for graphs generated by $RRH(n, 0, m)$ for different values of n and m.

a graph has a reduction sequence, i.e., it is solvable by any SR-Planner, then it can be generated by the procedure RRH for $p = 1$.

□

5.5 Embedding over- and under-constrained subgraph

Despite that we focus on well-constrained graphs, we can easily generalize the algorithm RRH to model over-and under-constrained graphs. To embed under-constrained subgraphs in the generated graph, we can easily remove randomly chosen edges from the graph H generated in step 5 of the algorithm RRH. Adding edges between randomly selected pair of nodes from the graph H leads to an over-constrained graph.

5.6 Experimental results

We have successfully implemented the procedure RRH. We conducted experiments to evaluate the solvability of graphs generated by procedure RRH.

n	p									
	50	100	150	200	250	300	350	400	450	500
500	10	5	4	3	2	2	2	2	2	1
600	12	6	4	3	3	2	2	2	2	2
700	14	7	5	4	3	3	2	2	2	2
800	16	8	6	4	4	3	3	2	2	2
900	18	9	6	5	4	3	3	3	2	2
1000	20	10	7	5	5	4	3	3	3	2

TABLE 5.5: The average number of subgraphs detected for different graphs generated by $RRH(n, 0, m)$ and for different values of n and m. (the expected optimal case: using the procedure RH0 (see algorithm 8)to generate subgraphs by algorithm 9 step 5 ; and for decomposition, using the MM planner modified by pinning all edges, on in turn (section 4.5.4)

Each generated graph that has a known number of subgraphs will be decomposed to show how those subgraphs are detected. Experiments are done for graph size n in $\{500, 600, 700, 800, 900, 1000\}$ and for the largest subgraph size m in $\{50, 10.5, 150, 200, 250, 300, 350, 400, 450, 500\}$. To ensure that subgraphs are not decomposable, we have fixed the parameter p to 0 and the size m of the largestest subgraph to 50. We recall that for the other values of p and m, experiments are presented in section 5.3.

We calculate the mean of the number of subgraphs detected after the decomposition. Due to the randomness, for each size, we generate 20 instances of each situation and calculate the mean. Table 5.4 gives the average number of subgraphs detected after decomposition. Theoretically, we expect that the number of subgraphs detected is at least equal to n/m. Table 5.4 shows the opposite. For example, for $n = 500$ and $m = 50$, instead of detecting at least $500/50 = 10$ subgraphs, the decomposition returns only an average of 5.85. The method of decomposition that we used did not detect all subgraphs (see section 4.5.4). We note that For $RRH(500, 0, 500)$, the average was 1.1, this mean that in some cases when generating a graph using the procedure RRH (500,0) (step 5 of algorithm 9), the resulted graphs are not always non-decomposable due to randomness. Notice, those results may change with

others decomposition algorithm. Figure 5.9 shows a large graph composed of 500 vertices, generated by $RRH(500, 0, 50)$. It contains ten non-reducible subgraphs, each one is generated in step 5 of algorithm 9 and has a size of 50 nodes. To be differentiated, subgraphs are plotted in different colours. Some subgraphs will be merged because the creation process is a recursive replacement of edge of the graph by a new generated subgraph H (step 6 of algorithm 9.). In figure 5.9, the yellow subgraph is merged with the orange and the brown with light-blue. To find the best decomposition possible, a good planner must be able to isolate all of them. In table 5.4 (The top-left cell of the table), the average number of detected subgraphs is 5.85 but the optimal average is $500/10 = 10$, as defined by the parameters of the algorithm.

If the algorithm 9 uses the procedure RH0 of the algorithm 8 instead of the procedure RH of algorithm 7. And if the MM-Planner was used with the modified version (pin all edges, one one in turn) as described in section 4.5.4, then the expected optimal decomposition must be as defined by figure 5.5.

5.7 Conclusion

We presented two algorithms for generating $2D$ geometric constraint graphs. The first one, RH, can be used to produce non-decomposable graphs or graphs with a given degree of decomposability. The second algorithm, RRH, can serve as a generator of graphs with the desired size of the larger subgraph. It can also be parameterized to generate graphs that are solvable by SR-Planners or MM-planners. We conducted an experimental study to show how generated graphs are decomposable. Our graph generator is complete: it generates all the domain of well-constrained geometric systems; Fast: linear in the number of iterations; Customizable: it requires few parameters to generate a class-specific graph with desired properties. It can be used to test and observe

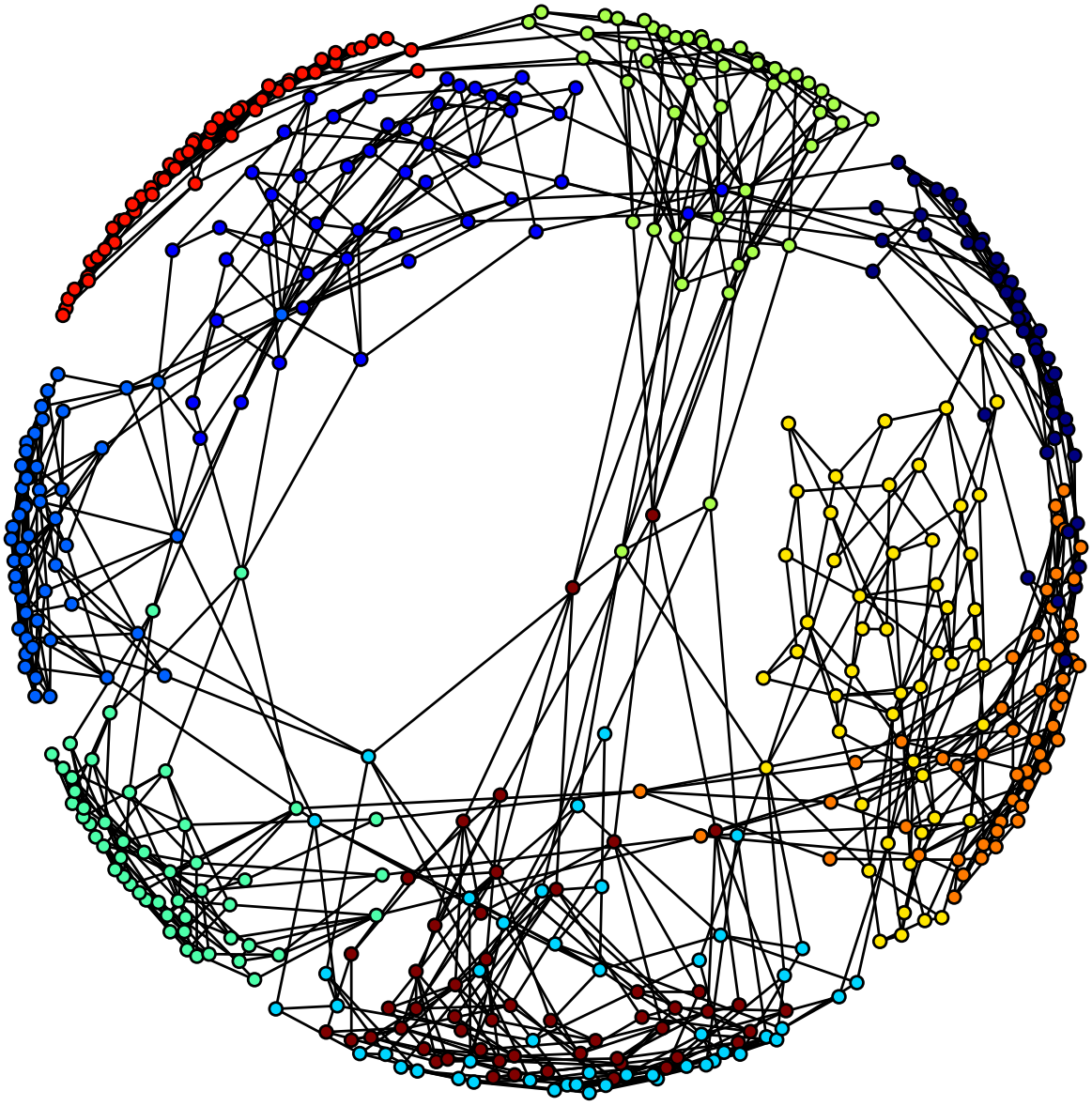


FIGURE 5.9: A well-constrained graph of 500 vertices generated by $RRH(500, 0, 50)$.

the behavior of many SR-planners or MM-planners. Moreover, it enables the comparison of solving methods with more data analysis rather than pure theory. Efficient and straightforward: based on strong theorems and simple to be implemented. It has been validated experimentally by decomposing generated graphs with a well-chosen planner.

Chapter 6

Conclusions and Future Work

Contents

6.1 Summary and Conclusions 100

6.2 Further works 102

6.1 Summary and Conclusions

The study was set out to explore the geometric constraint solving methods and the design of a geometric constraint graph generator that is very helpful for the analyzing the performance of geometric constraint solvers.

First, we have drawn on the powerful results from rigidity theory and combinatorial theory. We have shown that the geometric constraint problem is well understood in two dimensions; however, only partial similar results are available in three dimensions. We have then studied some solving approaches, particularly, numerical solvers. Those latter are $O(n^3)$ or worse and their acceleration consists in limiting their use to the smallest possible system of equations, which can be done by decomposing the geometric constraint problem using graph-based approaches. There are two categories of graph-based

methods: constructive solvers and general solvers. The first class that consists of finding all triangles of the geometric constraint graph is limited only to a sub-class of ruler and compass constructible GCS, and their generalization is tough. The second category is based on the computation of the flow or the maximum matching of the constraint graph with some appropriate connectivity analysis. This class is general, i.e., any GCS can be solved.

Major graph-based approaches are discussed or compared theoretically, without presenting any tests highlighting their performances. There is a great need for a tool which can help to perform tests with more data analysis rather than pure theory. Hence, we have designed our geometric constraint graph generator. In our design, we have taken care of many details, dictated by many results provided in the literature. Our starting point was to study the results provided by Henneberg [19] and discussed by Tay and Whiteley [77] . Those results are addressed to the structural topology area on the construction of rigid frameworks. This method gave us a solid starting base. We have adapted this method to the particular need of geometric constraints solving in CAD context.

First of all, the generator has to cover all the domain of geometric constraints solvers: without this property, it will be useless. This important point was achieved and theoretically proved. The second goal is how to generate a particular class of graphs. Our study was based on the classification resumed in the work of Hoffman et al. [15] , particularly the two main types of geometric constraint problems, the constructive and the general ones. Constructive solvers are well studied, and cover perhaps the majority of cases presented in the real CAD software. The second class is the general solvers, and they handle any geometric constraint problem. Another challenge in our generator design is the creation of non-decomposable geometric constraint graphs. As far as we know, no such method was proposed before. We have also introduced two key parameters that represent the decomposability metrics, the size of the

largest subgraph and the degree of decomposability. These two parameters can decide how fast will be the resolution process.

We have proposed two procedures. The first one is $RH(n, p)$, it produces a graph G of size n with a degree of decomposability equal to p. A value of $p = 0$, mean that the graph is generally non-decomposable for size greater than 40 vertex. The second procedure, $RRH(n, p, m)$, generates a graph G of size n with a number of subgraphs equal to m and the size of the largest subgraph equal to m. When $p = 1$, RRH generates only graphs that are decomposable by SR-Planners, more than that, we have proved that it cover all the domain of this class. Otherwise,i.e. if $p < 1$ it generates graphs decomposable by MM-Planners. We think that those two procedures are mostly sufficient. Finally, we believe that all required properties of the generator were achieved. The remaining step is to test how a geometric constraint solver perform when taking generated graphs as input. We have implemented a well-chosen general geometric constraint solver proposed by Ait-Aoudia et al. [10]. Several experiments was reported in this thesis, the results were as expected, they consolidate our theoretical findings and motivate its practical usage. We can conclude that the proposed generator is ready to be used for the performance analysis of any geometric constraint solver.

6.2 Further works

There are several aspect where there is potential for more study in the future. We recommend and suggest that further development should be undertaken in the following aspects:

- Extending the proposed generator to produce graphs in 3D space. It is also useful to study the case where geometric elements have more than two DOFs and geometric constraint can eliminate more than one DOF.

- It is very interesting to evaluate some well discussed graph-based methods presented in the literature using our graph generator. A comparison of the proposed methods can be made with more data analysis rather than pure theory.

Appendix A

Some samples of non-decomposable graphs

In this section, we present some samples of non-decomposable graphs with different sizes. The complete list of graphs with size ranging from 6 to 100 can be downloaded from [80].

- **Number of nodes = 6**
  Number of edges = 9
  V = {1, ...,6}
  E = { (1, 5), (1, 4), (1, 6), (3, 5), (3, 4), (3, 6), (2, 5), (2, 4), (2, 6) }

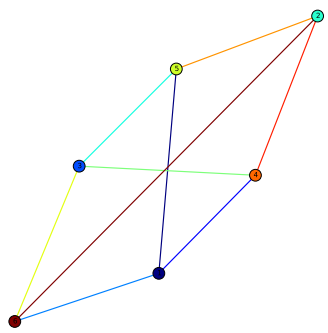


FIGURE A.1: A non-decomposable graph with 6 nodes.

- **Number of nodes = 8**
  Number of edges = 13
  V = {1, ...,8}
  E = { (1, 3), (1, 2), (1, 6), (3, 5), (3, 4), (2, 8), (2, 7), (5, 8), (5, 7), (5, 6), (4, 8), (4, 7), (4, 6) }

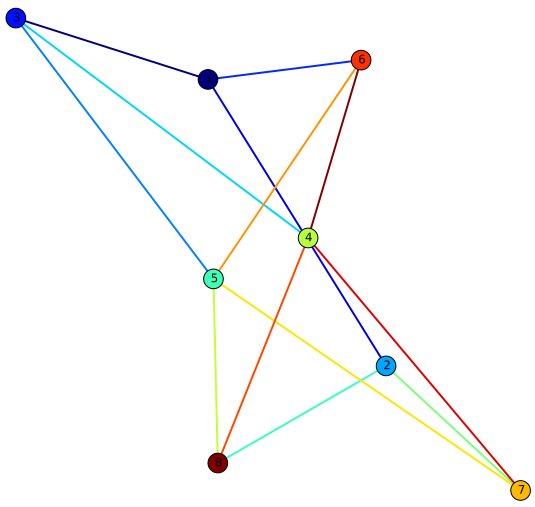


FIGURE A.2: A non-decomposable graph with 8 nodes.

- **Number of nodes = 9**
  Number of edges = 15
  V = {1, ...,9}
  E = { (1, 9), (1, 4), (1, 7), (3, 5), (3, 7), (3, 6), (2, 5), (2, 4), (2, 7), (2, 6), (5, 9), (5, 8), (4, 8), (7, 8), (6, 9) }

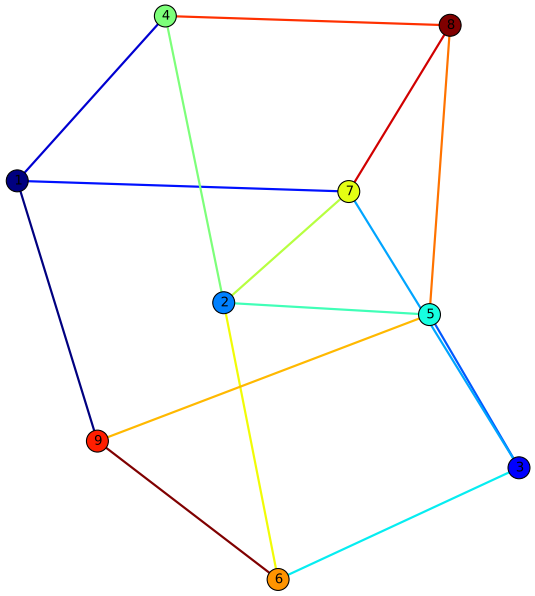


FIGURE A.3: A non-decomposable graph with 9 nodes.

- **Number of nodes = 10**
  Number of edges = 17
  V = {1, ...,10}
  E = { (10, 8), (10, 3), (10, 2), (1, 9), (1, 3), (1, 4), (1, 7), (3, 5), (3, 6), (2, 9), (2, 5), (2, 4), (5, 8), (5, 7), (4, 6), (7, 6), (9, 8) }

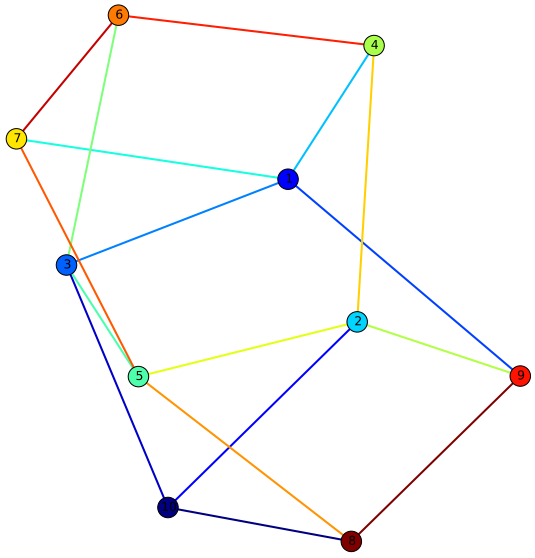


FIGURE A.4: A non-decomposable graph with 10 nodes.

- **Number of nodes = 20**
  Number of edges = 37
  V = {1, ...,20}
  E = { (11, 3), (11, 12), (11, 6), (10, 1), (10, 3), (10, 12), (10, 15), (13, 19), (13, 12), (13, 17), (15, 2), (15, 18), (14, 2), (14, 5), (14, 17), (14, 6), (17, 1), (16, 8), (16, 4), (16, 6), (19, 9), (19, 18), (18, 8), (18, 7), (20, 1), (20, 4), (20, 7), (1, 2), (1, 6), (1, 8), (3, 5), (3, 7), (2, 9), (2, 7), (5, 8), (5, 4), (9, 8) }

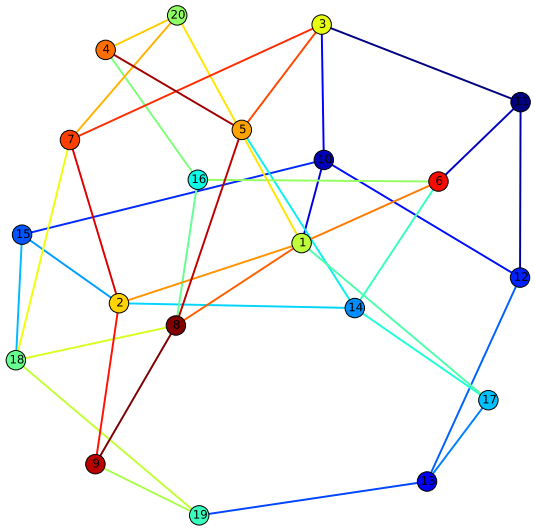


FIGURE A.5: A non-decomposable graph with 20 nodes.

- **Number of nodes = 30**
  Number of edges = 57
  V = {1, ...,30} E = { (24, 1), (24, 9), (24, 29), (24, 23), (25, 19), (25, 5), (25, 16), (26, 18), (26, 7), (26, 10), (27, 13), (27, 12), (27, 28), (27, 4), (20, 12), (20, 29), (20, 6), (21, 13), (21, 5), (21, 23), (22, 3), (22, 30), (22, 7), (22, 23), (23, 4), (28, 5), (28, 7), (29, 16), (1, 10), (1, 3), (1, 15), (3, 2), (3, 5), (2, 8), (2, 16), (2, 18), (5, 10), (4, 11), (4, 14), (4, 17), (4, 8), (7, 6), (6, 11), (6, 17), (9, 13), (9, 12), (8, 19), (11, 18), (11, 30), (11, 15), (13, 14), (13, 16), (12, 18), (12, 14), (15, 17), (14, 19), (14, 30) }

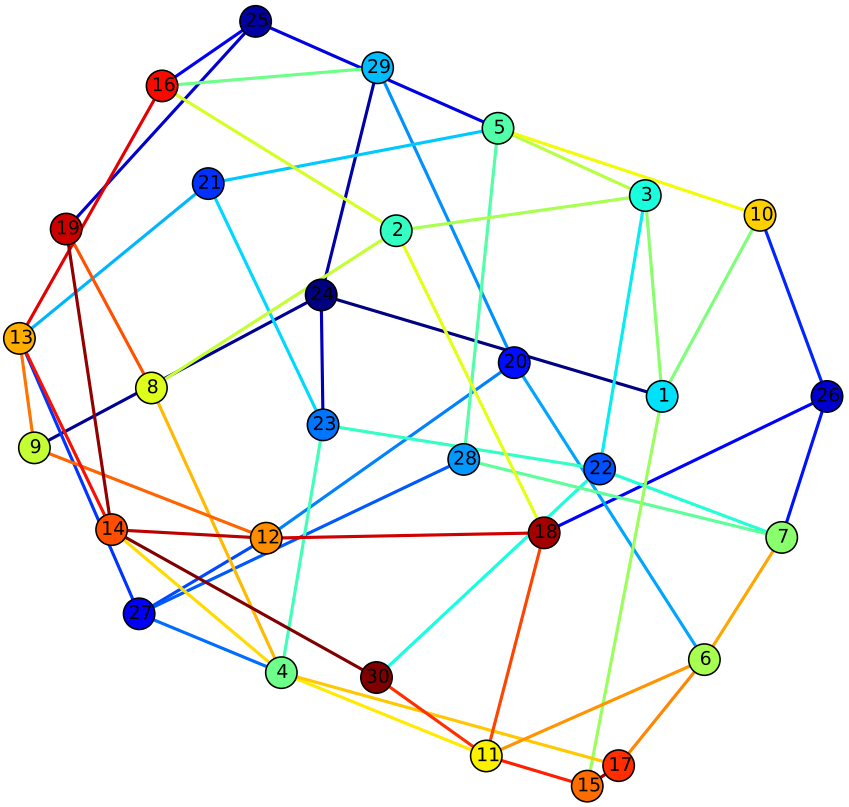


FIGURE A.6: A non-decomposable graph with 30 nodes.

- **Number of nodes = 40**
  Number of edges = 77
  V = {1, ...,40} E = { (24, 12), (24, 37), (24, 40), (24, 16), (25, 18), (25, 2), (25, 28), (25, 38), (26, 1), (26, 32), (26, 5), (26, 29), (27, 9), (27, 28), (27, 37), (27, 4), (20, 33), (20, 8), (20, 7), (20, 16), (21, 32), (21, 12), (21, 36), (21, 40), (22, 11), (22, 18), (22, 39), (22, 5), (22, 31), (23, 10), (23, 3), (23, 31), (28, 1), (29, 9), (29, 35), (29, 6), (40, 17), (1, 11), (1, 18), (1, 9), (1, 8), (3, 11), (3, 39), (3, 15), (2, 19), (2, 39), (2, 30), (2, 14), (5, 19), (5, 13), (5, 34), (5, 6), (4, 10), (4, 13), (4, 14), (4, 32), (4, 8), (7, 10), (7, 30), (7, 6), (6, 33), (6, 38), (9, 15), (9, 17), (9, 33), (8, 15), (8, 17), (13, 35), (38, 30), (11, 34), (12, 35), (12, 34), (15, 31), (15, 16), (14, 36), (16, 19), (37, 36) }

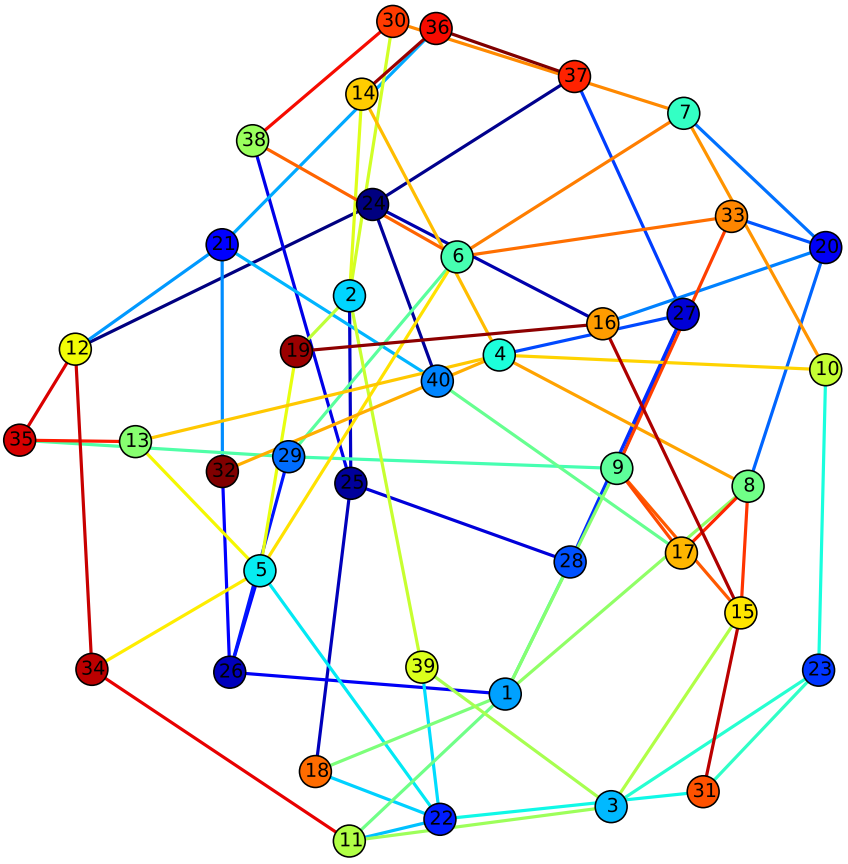


FIGURE A.7: A non-decomposable graph with 40 nodes.

- **Number of nodes = 50**
  Number of edges = 97
  V = {1, ...,50}
  E = { (42, 39), (42, 38), (42, 15), (48, 9), (48, 13), (48, 16), (43, 39), (43, 31), (43, 41), (23, 1), (23, 18), (23, 7), (24, 9), (24, 18), (24, 40), (25, 11), (25, 27), (25, 37), (26, 2), (26, 8), (26, 30), (27, 33), (27, 19), (20, 2), (20, 7), (20, 6), (21, 32), (21, 31), (21, 49), (21, 10), (22, 46), (22, 47), (22, 1), (22, 15), (49, 33), (49, 1), (46, 12), (46, 37), (47, 18), (47, 37), (44, 13), (44, 14), (44, 34), (45, 9), (45, 10), (45, 34), (28, 10), (28, 14), (28, 4), (29, 3), (29, 2), (29, 16), (40, 17), (40, 16), (41, 32), (41, 6), (41, 17), (41, 50), (1, 39), (1, 17), (1, 6), (1, 9), (3, 34), (3, 14), (3, 50), (2, 17), (2, 4), (5, 11), (5, 14), (5, 6), (4, 19), (4, 35), (4, 16), (7, 8), (7, 13), (6, 13), (6, 38), (9, 35), (8, 12), (8, 31), (8, 50), (39, 16), (38, 33), (11, 33), (11, 17), (10, 13), (10, 12), (10, 30), (13, 36), (12, 36), (12, 34), (15, 31), (14, 32), (19, 30), (19, 37), (19, 18), (36, 35) }

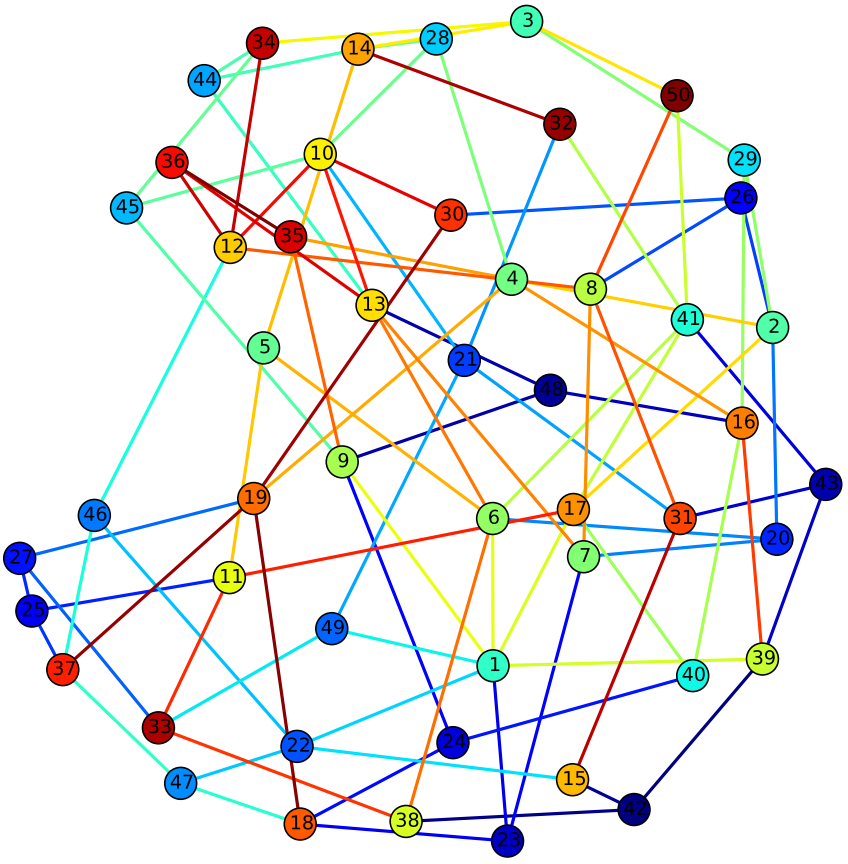


FIGURE A.8: A non-decomposable graph with 50 nodes.

- **Number of nodes = 60**
  Number of edges = 117
  V = {1, ...,60}
  E = { (56, 25), (56, 26), (56, 52), (28, 24), (28, 18), (28, 6), (45, 18), (45, 6), (45, 8), (50, 60), (50, 25), (50, 1), (50, 5), (60, 11), (60, 20), (53, 10), (53, 39), (53, 4), (34, 11), (34, 32), (34, 40), (34, 52), (23, 1), (23, 2), (23, 37), (24, 1), (24, 8), (24, 14), (25, 10), (25, 58), (25, 47), (26, 27), (26, 31), (26, 4), (27, 9), (27, 16), (20, 9), (20, 13), (20, 35), (21, 9), (21, 17), (21, 22), (22, 10), (22, 5), (49, 36), (49, 17), (49, 43), (46, 2), (46, 31), (46, 30), (47, 31), (47, 8), (44, 11), (44, 5), (44, 48), (48, 55), (48, 32), (48, 6), (42, 39), (42, 36), (42, 4), (29, 10), (29, 57), (29, 15), (29, 58), (40, 57), (40, 7), (41, 13), (41, 38), (41, 59), (1, 13), (1, 16), (1, 6), (3, 11), (3, 4), (3, 17), (3, 14), (2, 16), (5, 12), (5, 59), (5, 19), (5, 9), (4, 43), (4, 7), (7, 9), (7, 55), (7, 14), (6, 58), (6, 51), (6, 37), (9, 15), (9, 33), (8, 12), (33, 18), (33, 31), (43, 57), (43, 16), (39, 37), (39, 38), (12, 32), (59, 32), (32, 10), (11, 31), (11, 51), (10, 37), (13, 55), (38, 17), (15, 18), (15, 14), (16, 54), (16, 52), (19, 30), (19, 36), (18, 54), (18, 35), (30, 35), (51, 54) }

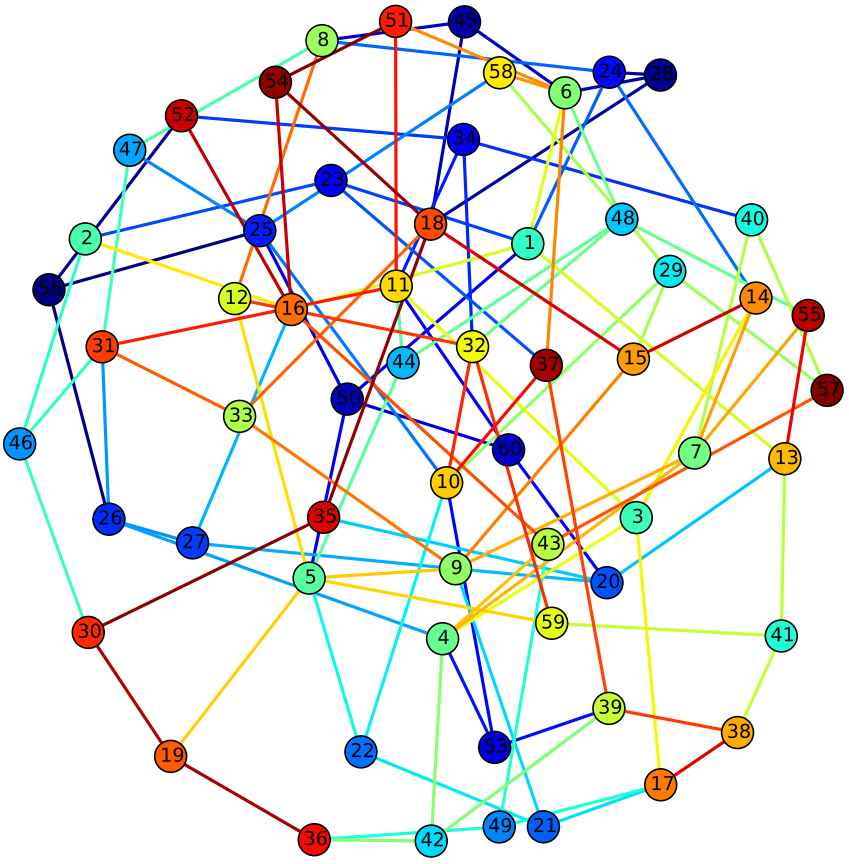


FIGURE A.9: A non-decomposable graph with 60 nodes.

- **Number of nodes = 70**
  Number of edges = 137
  V = {1, ...,70}
  E = { (56, 1), (56, 11), (56, 45), (42, 11), (42, 32), (42, 17), (22, 39), (22, 44), (22, 53), (22, 3), (50, 18), (50, 43), (50, 48), (60, 19), (60, 18), (60, 15), (61, 24), (61, 5), (61, 6), (62, 3), (62, 63), (62, 45), (62, 41), (63, 55), (63, 47), (63, 64), (64, 5), (64, 29), (65, 11), (65, 6), (65, 16), (66, 8), (66, 15), (66, 4), (67, 2), (67, 12), (67, 4), (68, 32), (68, 20), (68, 52), (69, 2), (69, 45), (69, 14), (52, 43), (52, 4), (23, 37), (23, 53), (23, 7), (24, 19), (24, 1), (25, 11), (25, 3), (25, 16), (25, 41), (26, 1), (26, 10), (26, 9), (27, 3), (27, 43), (27, 7), (20, 12), (20, 46), (20, 43), (20, 6), (20, 8), (21, 2), (21, 4), (21, 48), (21, 49), (48, 11), (49, 47), (49, 45), (46, 15), (46, 57), (47, 17), (44, 28), (44, 6), (45, 30), (45, 38), (45, 70), (28, 33), (28, 34), (28, 4), (28, 41), (43, 2), (40, 33), (40, 35), (40, 16), (41, 10), (1, 4), (1, 14), (35, 18), (35, 3), (3, 17), (3, 12), (3, 58), (5, 17), (5, 16), (5, 31), (5, 37), (4, 19), (7, 39), (7, 13), (6, 10), (6, 13), (9, 33), (9, 51), (9, 13), (8, 55), (8, 37), (8, 70), (13, 15), (13, 29), (38, 18), (38, 29), (70, 36), (59, 19), (59, 31), (59, 15), (29, 51), (32, 12), (58, 51), (58, 39), (10, 33), (39, 14), (12, 30), (15, 54), (15, 30), (15, 34), (55, 34), (19, 54), (54, 53),

(31, 36), (31, 34), (36, 57), (53, 34), (34, 57) }

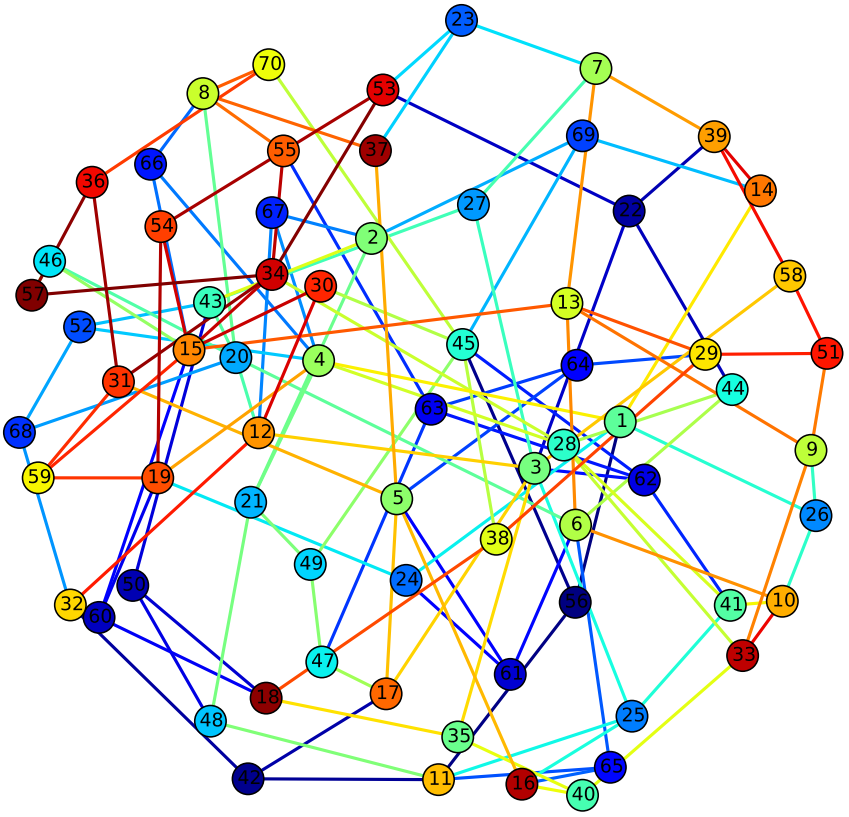


FIGURE A.10: A non-decomposable graph with 70 nodes.

- **Number of nodes = 80**
  Number of edges = 157
  V = {1, ...,80}
  E = { (30, 15), (30, 46), (30, 50), (30, 1), (30, 55), (30, 63), (30, 79), (28, 68), (28, 36), (28, 23), (48, 5), (48, 38), (48, 20), (29, 69), (29, 26), (29, 2), (29, 37), (29, 66), (60, 31), (60, 45), (60, 6), (61, 46), (61, 13), (61, 43), (62, 1), (62, 38), (62, 65), (62, 6), (63, 39), (63, 51), (63, 67), (64, 54), (64, 44), (64, 43), (65, 27), (65, 31), (66, 25), (66, 36), (67, 27), (67, 6), (68, 77), (68, 44), (68, 71), (69, 46), (69, 18), (80, 19), (80, 79), (80, 17), (32, 47), (32, 13), (32, 74), (32, 18), (24, 8), (24, 14), (24, 7), (25, 10), (25, 2), (25, 5), (25, 58), (26, 19), (26, 17), (26, 53), (27, 18), (27, 52), (27, 75), (27, 47), (27, 78), (20, 9), (20, 51), (20, 40), (20, 23), (58, 55), (58, 73), (58, 76), (22, 19), (22, 12), (22, 14), (23, 15), (46, 75), (47, 21), (44, 73), (44, 38), (44, 35), (45, 33), (45, 9), (42, 11), (42, 43), (42, 14), (43, 5), (40, 2), (40, 7), (41, 1), (41, 8), (41, 12), (41, 71), (1, 11), (1, 13), (1, 2), (35, 74), (35, 31), (3, 31), (3, 38), (3, 73), (3, 4), (2, 6), (5, 18), (5, 57), (5, 56), (5, 9), (4, 39), (4, 15), (4, 14), (4, 55), (4, 6), (7, 17), (7, 50), (6, 12), (6, 33), (6, 19), (9, 53), (9, 49), (8, 56), (8, 37),

(49, 31), (49, 70), (49, 71), (49, 78), (39, 12), (39, 59), (34, 55), (34, 10), (34, 72), (77, 33), (77, 12), (76, 21), (76, 52), (75, 72), (74, 54), (72, 19), (70, 15), (70, 17), (59, 33), (59, 79), (78, 56), (11, 16), (10, 15), (10, 21), (13, 16), (13, 54), (13, 53), (12, 37), (14, 36), (16, 51), (16, 21), (54, 57), (57, 52), (18, 50) }

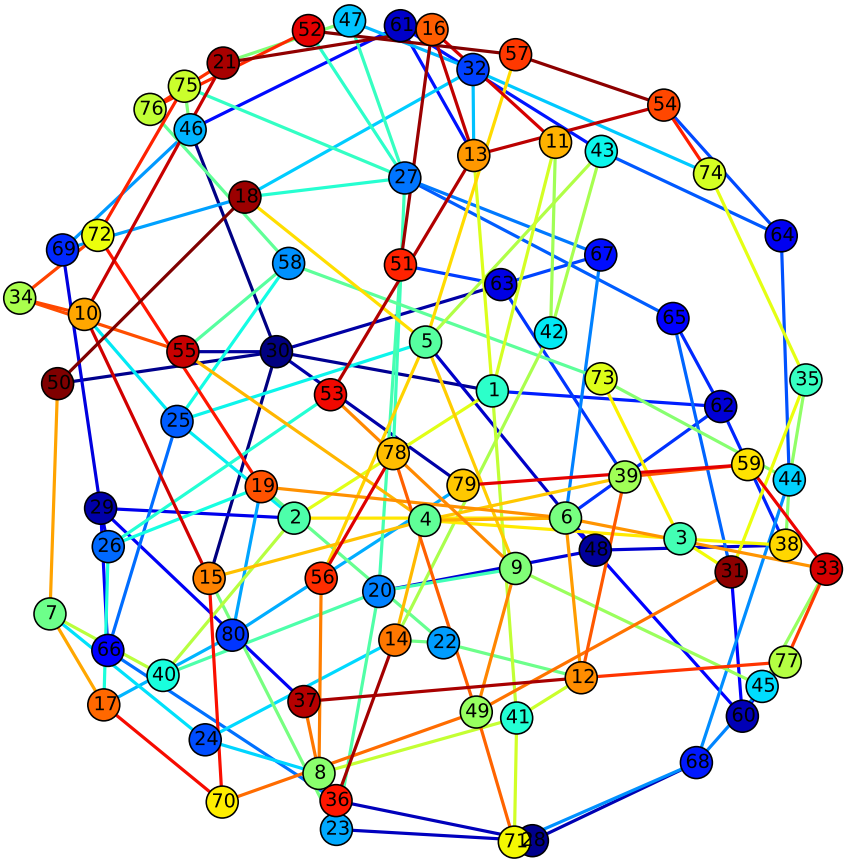


FIGURE A.11: A non-decomposable graph with 80 nodes.

- **Number of nodes = 90**
  Number of edges = 177
  V = {1, ...,90}
  E = { (24, 12), (24, 56), (24, 43), (24, 35), (24, 41), (24, 40), (24, 7), (24, 81), (25, 59), (25, 51), (25, 72), (25, 68), (25, 69), (25, 87), (26, 13), (26, 2), (26, 17), (27, 80), (27, 63), (27, 21), (27, 16), (20, 47), (20, 44), (20, 89), (20, 50), (21, 60), (21, 40), (21, 77), (21, 73), (21, 4), (22, 83), (22, 30), (22, 4), (22, 85), (23, 2), (23, 5), (23, 43), (23, 78), (28, 45), (28, 15), (28, 17), (28, 41), (29, 60), (29, 39), (29, 30), (4, 10), (4, 5), (4, 6), (4, 14), (8, 11), (8, 89), (8, 72), (8, 9), (8, 81), (8, 85), (59, 9), (59, 2), (59, 65), (59, 74), (58, 33), (58, 18), (58, 67), (55, 1), (55, 64), (55, 87), (54, 33), (54, 11), (54, 62), (54,

38), (57, 9), (57, 12), (57, 66), (56, 90), (56, 62), (56, 71), (56, 84), (51, 73), (51, 17), (50, 33), (50, 61), (53, 44), (53, 37), (53, 6), (52, 16), (52, 40), (52, 70), (88, 32), (88, 31), (88, 12), (89, 39), (82, 1), (82, 31), (82, 13), (83, 13), (83, 45), (80, 2), (80, 43), (81, 39), (86, 47), (86, 71), (86, 67), (87, 62), (84, 18), (84, 48), (85, 69), (3, 46), (3, 2), (3, 17), (3, 70), (7, 46), (7, 42), (7, 79), (7, 11), (39, 31), (39, 14), (38, 68), (38, 36), (38, 90), (33, 76), (33, 73), (32, 47), (32, 35), (32, 34), (31, 63), (31, 78), (30, 6), (37, 11), (37, 5), (36, 12), (36, 14), (35, 77), (34, 5), (34, 48), (60, 65), (60, 17), (61, 47), (61, 16), (63, 11), (64, 65), (64, 41), (65, 75), (66, 15), (66, 71), (67, 79), (68, 19), (69, 6), (2, 19), (2, 45), (6, 15), (6, 19), (6, 18), (6, 1), (6, 76), (90, 10), (11, 16), (11, 41), (10, 46), (10, 13), (10, 18), (12, 79), (15, 44), (15, 75), (14, 49), (17, 78), (16, 19), (48, 9), (48, 75), (49, 1), (49, 43), (47, 76), (47, 77), (47, 70), (42, 74), (42, 72), (40, 74), (40, 71), (1, 9) }

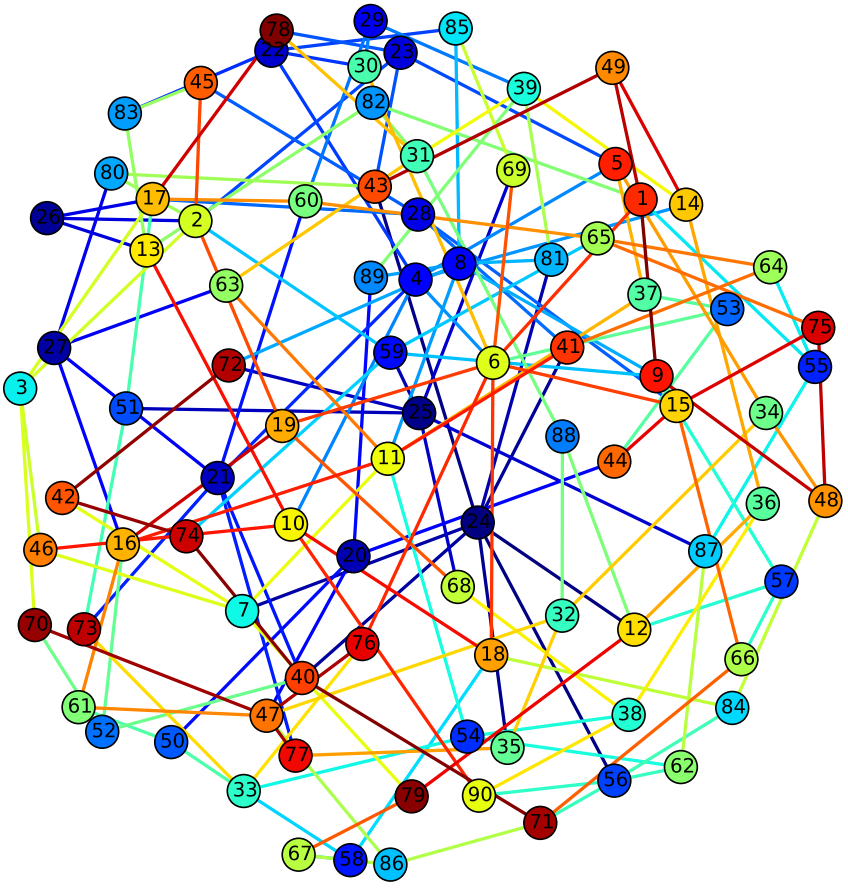


FIGURE A.12: A non-decomposable graph with 90 nodes.

- **Number of nodes = 100**
  Number of edges = 197
  V = {1, ...,100}

E = { (24, 57), (24, 22), (24, 70), (25, 47), (25, 84), (25, 58), (25, 22), (25, 69), (26, 1), (26, 100), (26, 42), (26, 71), (27, 44), (27, 2), (27, 89), (27, 40), (27, 74), (20, 11), (20, 76), (20, 88), (20, 5), (21, 14), (21, 23), (21, 51), (21, 1), (21, 88), (21, 68), (22, 59), (22, 17), (22, 23), (22, 52), (23, 50), (23, 39), (23, 44), (28, 45), (28, 4), (28, 66), (28, 36), (29, 99), (29, 90), (29, 12), (29, 37), (29, 10), (4, 49), (4, 52), (4, 1), (4, 6), (4, 81), (8, 52), (8, 96), (8, 70), (8, 34), (59, 80), (59, 62), (58, 39), (58, 17), (55, 80), (55, 63), (55, 15), (55, 36), (54, 77), (54, 9), (54, 92), (54, 71), (54, 70), (57, 69), (57, 93), (57, 48), (56, 48), (56, 34), (56, 73), (56, 68), (56, 92), (56, 97), (56, 96), (51, 32), (51, 85), (51, 53), (51, 48), (50, 82), (50, 75), (50, 6), (53, 11), (53, 31), (53, 63), (88, 67), (89, 37), (89, 71), (82, 68), (82, 63), (83, 69), (83, 38), (83, 66), (80, 100), (80, 75), (81, 45), (81, 41), (86, 32), (86, 12), (86, 37), (86, 87), (87, 39), (87, 43), (87, 94), (84, 39), (84, 7), (85, 62), (85, 97), (3, 13), (3, 73), (3, 31), (7, 91), (7, 9), (7, 5), (100, 45), (39, 15), (39, 72), (38, 36), (38, 76), (33, 47), (33, 93), (33, 5), (33, 40), (33, 17), (32, 14), (32, 78), (31, 61), (31, 17), (30, 12), (30, 14), (30, 35), (30, 60), (30, 91), (30, 69), (37, 13), (35, 1), (35, 65), (34, 46), (34, 62), (34, 65), (60, 64), (60, 40), (61, 98), (61, 45), (61, 16), (63, 64), (64, 6), (65, 48), (66, 90), (67, 9), (67, 12), (69, 77), (69, 97), (2, 46), (2, 10), (2, 12), (6, 10), (6, 13), (6, 14), (6, 98), (99, 47), (99, 78), (98, 42), (91, 18), (90, 49), (93, 15), (92, 95), (95, 76), (95, 42), (94, 19), (94, 9), (96, 47), (11, 13), (10, 19), (13, 41), (12, 41), (15, 43), (15, 74), (15, 9), (14, 9), (16, 9), (16, 75), (16, 5), (19, 18), (18, 74), (18, 79), (49, 9), (49, 5), (46, 79), (44, 79), (45, 75), (42, 5), (43, 5), (1, 73), (9, 72), (77, 72), (71, 78) }

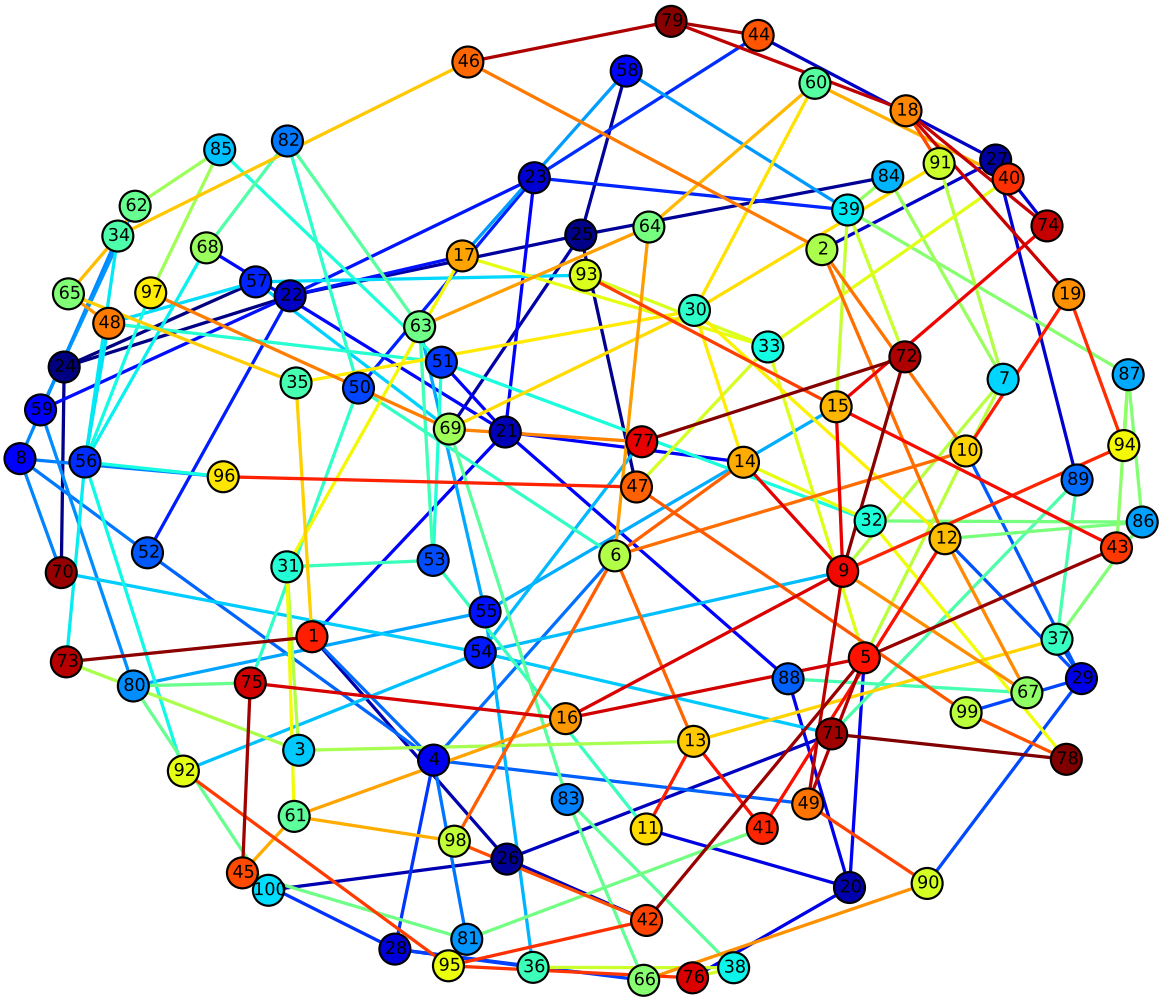


FIGURE A.13: A non-decomposable graph with 100 nodes.

Bibliography

[1] Adel Moussaoui and Samy Ait-Aoudia. Generator of 2d geometric constraint graphs. *Computer-Aided Design and Applications*, 13(03): 271–280, 2016. URL http://dx.doi.org/10.1080/16864360.2015.1114384.

[2] Christoph M Hoffmann and Robert Joan-Arinyo. A brief on constraint solving. *Computer-Aided Design and Applications*, 2(5):655–663, 2005. URL http://dx.doi.org/10.1080/16864360.2005.10738330.

[3] Yuan Zhang, Shutang Liu, Xiuyang Zhao, and Zhongtian Jia. Theoretic analysis of unique localization for wireless sensor networks. *Ad Hoc Networks*, 10(3):623–634, 2012. URL http://dx.doi.org/10.1016/j.adhoc.2011.06.016.

[4] Leo Liberti, Carlile Lavor, Nelson Maculan, and Antonio Mucherino. Euclidean distance geometry and applications. *SIAM Review*, 56(1):3–69, 2014. URL http://dx.doi.org/10.1137/120875909.

[5] Bruce Hendrickson. Conditions for unique graph realizations. *SIAM Journal on Computing*, 21(1):65–84, 1992. URL http://dx.doi.org/10.1137/0221008.

[6] James B Saxe. Embeddability of weighted graphs in k-space is strongly np-hard. In *Proceedings of 17th Allerton Conference in Communications, Control and Computing*, pages 480–489, 1979.

[7] Bernhard Bettig and Christoph M Hoffmann. Geometric constraint solving in parametric computer-aided design. *Journal of computing and information science in engineering*, 11(2):021001, 2011. URL http://dx.doi.org/10.1115/1.3593408.

[8] Christophe Jermann, Gilles Trombettoni, Bertrand Neveu, and Pascal Mathis. Decomposition of geometric constraint systems: a survey. *International Journal of Computational Geometry & Applications*, 16(05n06):379–414, 2006. URL http://dx.doi.org/10.1142/S0218195906002105.

[9] Samy Ait-Aoudia and Sebti Foufou. A 2d geometric constraint solver using a graph reduction method. *Advances in Engineering Software*, 41(10): 1187–1194, 2010. URL http://dx.doi.org/10.1016/j.advengsoft.2010.07.008.

[10] Samy Ait-Aoudia, Roland Jegou, and Dominique Michelucci. Reduction of constraint systems. In *Proceedings of Compugraphics'93 Alvor, Algarve, Portugal*, pages 331–340. ACM, 1993.

[11] Ioannis Fudos and Christoph M Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics (TOG)*, 16(2):179–216, 1997. URL http://dx.doi.org/10.1145/248210.248223.

[12] Christoph M Hoffman, Andrew Lomonosov, and Meera Sitharam. Decomposition plans for geometric constraint problems, part ii: new algorithms. *Journal of Symbolic Computation*, 31(4):409–427, 2001. URL http://dx.doi.org/10.1006/jsco.2000.0403.

[13] Richard S Latham and Alan E Middleditch. Connectivity analysis: a tool for processing geometric constraints. *Computer-Aided Design*, 28

(11):917–928, 1996. URL http://dx.doi.org/10.1016/0010-4485(96)00023-1.

[14] John C Owen. Algebraic solution for geometry from dimensional constraints. In *Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*, pages 397–407. ACM, 1991. URL http://dx.doi.org/10.1145/112515.112573.

[15] Christoph M Hoffman, Andrew Lomonosov, and Meera Sitharam. Decomposition plans for geometric constraint systems, part i: Performance measures for CAD. *Journal of Symbolic Computation*, 31(4):367–408, 2001. URL http://dx.doi.org/10.1006/jsco.2000.0402.

[16] Robert Joan-Arinyo, Marta Tarrés-Puertas, and Sebastià Vila-Marta. Decomposition of geometric constraint graphs based on computing fundamental circuits. correctness and complexity. *Computer-Aided Design*, 52: 1–16, 2014. URL http://dx.doi.org/10.1016/j.cad.2014.02.006.

[17] Robert Joan-Arinyo, Antoni Soto-Riera, Sebastià Vila-Marta, and Josep Vilaplana-Pasto. Revisiting decomposition analysis of geometric constraint graphs. *Computer-Aided Design*, 36(2):123–140, 2004. URL http://dx.doi.org/10.1016/S0010-4485(03)00057-5.

[18] Gerard Laman. On graphs and rigidity of plane skeletal structures. *Journal of Engineering mathematics*, 4(4):331–340, 1970. URL http://dx.doi.org/10.1007/bf01534980.

[19] Lebrecht Henneberg. *Die graphische Statik der starren Systeme*, volume 31. BG Teubner, 1911.

[20] Krishnaiyan Thulasiraman and Madisetti NS Swamy. *Graphs: theory and algorithms*. John Wiley & Sons, 2011. URL http://dx.doi.org/10.1002/9781118033104.

[21] Andrew L Dulmage and Nathan S Mendelsohn. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 10(4):516–534, 1958. URL http://dx.doi.org/10.4153/cjm-1958-052-0.

[22] Adel Moussaoui. *Modélisation géometrique par contraintes*. Mémoire de Magister, ESI, Ecole Nationale Supérieure en Informatique, 2000.

[23] Ben Roth. Rigid and flexible frameworks. *American Mathematical Monthly*, pages 6–21, 1981. URL http://dx.doi.org/10.2307/2320705.

[24] Hiroshi Imai. On combinatorial structures of line drawings of polyhedra. *Discrete Applied Mathematics*, 10(1):79–92, 1985. URL http://dx.doi.org/10.1016/0166-218x(85)90060-5.

[25] Donald J Jacobs and Bruce Hendrickson. An algorithm for two-dimensional rigidity percolation: the pebble game. *Journal of Computational Physics*, 137(2):346–365, 1997. URL http://dx.doi.org/10.1006/jcph.1997.5809.

[26] Dominique Michelucci and Sebti Foufou. Geometric constraint solving: The witness configuration method. *Computer-Aided Design*, 38(4):284–299, 2006. URL http://dx.doi.org/10.1016/j.cad.2006.01.005.

[27] Dominique Michelucci and Sebti Foufou. Interrogating witnesses for geometric constraint solving. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 343–348. ACM, 2009. URL http://dx.doi.org/10.1145/1629255.1629301.

[28] Yong Zhou. *Combinatorial decomposition, generic independence and algebraic complexity of geometric constraints systems: applications in biology and engineering*. PhD thesis, University of Florida, 2006.

[29] Christoph M Hoffmann and Ching-Shoei Chiang. Variable-radius circles of cluster merging in geometric constraints: I. translational clusters. *Computer-Aided Design*, 34(11):787–797, 2002. URL http://dx.doi.org/10.1016/s0010-4485(01)00139-7.

[30] Ching-Shoei Chiang and Robert Joan-Arinyo. Revisiting variable radius circles in constructive geometric constraint solving. *Computer aided geometric design*, 21(4):371–399, 2004. URL http://dx.doi.org/10.1016/j.cagd.2004.01.003.

[31] Guoqiang Mao, Barış Fidan, and Brian DO Anderson. Wireless sensor network localization techniques. *Computer networks*, 51(10):2529–2553, 2007. URL http://dx.doi.org/10.1016/j.comnet.2006.11.018.

[32] Adnan Sljoka. *Algorithms in rigidity theory with applications to protein flexibility and mechanical linkages*. Phd thesis,York University, 2012.

[33] Donald J Jacobs, Andrew J Rader, Leslie A Kuhn, and Michael F Thorpe. Protein flexibility predictions using graph theory. *Proteins: Structure, Function, and Bioinformatics*, 44(2):150–165, 2001. URL http://dx.doi.org/10.1002/prot.1081.

[34] Luis C González, Hui Wang, Dennis R Livesay, and Donald J Jacobs. A virtual pebble game to ensemble average graph rigidity. *Algorithms for molecuar biologie*, 2015. URL http://dx.doi.org/10.1186/s13015-015-0039-3.

[35] Harald Winroth. *Dynamic projective geometry*. Phd thesis, Tekniska högsk., 1999.

[36] Colette Laborde, Chronis Kynigos, Karen Hollebrands, and Rudolf Strässer. Teaching and learning geometry with technology. *Handbook of research on the psychology of mathematics education: Past, present and future*, pages 275–304, 2006.

[37] Markus Hohenwarter and Judith Preiner. Dynamic mathematics with GeoGebra. *AMC*, 10:12, 2007.

[38] Marc Freixas, Robert Joan-Arinyo, and Antoni Soto-Riera. A constraint-based dynamic geometry system. *Computer-Aided Design*, 42(2):151–161, 2010. URL http://dx.doi.org/10.1016/j.cad.2009.02.016.

[39] Martin H Gutknecht. A brief introduction to krylov space methods for solving linear systems. In *Frontiers of Computational Science*, pages 53–62. Springer, 2007. URL http://dx.doi.org/10.1007/978-3-540-46375-7_5.

[40] Michael T.. Heath. *Scientific Computing: An Introductory Survey*. McGraw-Hill, 1997. URL http://dx.doi.org/10.1007/s00897990373a.

[41] Michel Bierlaire. *Optimization: Principles and Algorithms*. EPFL Press, Lausanne, 2015. URL http://www.optimizationprinciplesalgorithms.com/.

[42] Samy Ait-Aoudia, Brahim Hamid, Adel Moussaoui, and Toufik Saadi. Solving geometric constraints by bissection. In *1st Mediterranean international conference on computer technologies*, pages 223–232, 1999.

[43] S Ait-Aoudia and A Moussaoui. Cogemo: A constraint-based geometric modeller. In *Swiss Conference on CAD/CAM Systems,(Neuchâtel, Swiss)*, 1999.

[44] R Baker Kearfott. Some tests of generalized bisection. *ACM Transactions on Mathematical Software (TOMS)*, 13(3):197–220, 1987. URL http://dx.doi.org/10.1145/29380.29862.

[45] RE Moore and L Qi. A successive interval test for nonlinear systems. *SIAM Journal on Numerical Analysis*, 19(4):845–850, 1982. URL http://dx.doi.org/10.1137/0719060.

[46] Rudolf Krawczyk and Arnold Neumaier. Interval slopes for rational functions and associated centered forms. *SIAM Journal on Numerical Analysis*, 22(3):604–616, 1985. URL http://dx.doi.org/10.1137/0722037.

[47] Samy Ait-Aoudia and Imad Mana. Numerical solving of geometric constraints by bisection: a distributed approach. *International Journal of Computing & Information Sciences*, 2(2):66, 2004. URL http://dx.doi.org/10.1109/iv.2002.1028766.

[48] H. Lamure and D. Michelucci. Solving geometric constraints by homotopy. *IEEE Trans. Visual. Comput. Graphics*, 2(1):28–34, mar 1996. doi: 10.1109/2945.489384. URL http://dx.doi.org/10.1109/2945.489384.

[49] Rémi Imbach. *Résolution de contraintes géométriques en guidant une méthode homotopique par la géométrie*. PhD thesis, Strasbourg, 2013.

[50] Aeron M Buchanan and Andrew W Fitzgibbon. Damped newton algorithms for matrix factorization with missing data. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 316–322. IEEE, 2005. URL http://dx.doi.org/10.1109/cvpr.2005.118.

[51] Glenn A Kramer. Using degrees of freedom analysis to solve geometric constraint systems. In *Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*, pages 371–378. ACM, 1991. URL http://dx.doi.org/10.1145/112515.112566.

[52] GA Kramer. Solving geometric constraint systems: a case study in kinematics mit press. *Cambridge, MA*, 1992.

[53] Ching-yao Hsu and Beat D Brüderlin. A hybrid constraint solver using exact and iterative geometric constructions. In *CAD Systems Development*, pages 265–279. Springer, 1997.

[54] Bjorn N Freeman-Benson, John Maloney, and Alan Borning. An incremental constraint solver. *Communications of the ACM*, 33(1):54–63, 1990. URL http://dx.doi.org/10.1145/76372.77531.

[55] Remco C Veltkamp and Farhad Arbab. Geometric constraint propagation with quantum labels. In *Computer Graphics and Mathematics*, pages 211–228. Springer, 1992. URL http://dx.doi.org/10.1007/978-3-642-77586-4_14.

[56] Alan Borning, Richard Anderson, and Bjorn Freeman-Benson. Indigo: A local propagation algorithm for inequality constraints. In *Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 129–136. ACM, 1996. URL http://dx.doi.org/10.1145/237091.237110.

[57] Bernd Aldefeld. Variation of geometries based on a geometric-reasoning method. *Computer-Aided Design*, 20(3):117–126, 1988. URL http://dx.doi.org/10.1016/0010-4485(88)90019-x.

[58] BD Brüderlin. Symbolic computer geometry for computer aided geometric design. *Advances in Design and Manufacturing Systems*, pages 177–181, 1990.

[59] Yasushi Yamaguchi and Fumihiko Kimura. A constraint modeling system for variational geometry. *Geometric Modeling for Product Engineering*, pages 221–233, 1990.

[60] Geir Sunde. A cad system with declarative specification of shape. In *Intelligent CAD Systems I*, pages 90–104. Springer, 1987. URL http://dx.doi.org/10.1007/978-3-642-72945-4_6.

[61] Anne Verroust, F Schonek, and Dieter Roller. Rule-oriented method for parameterized computer-aided design. *Computer-Aided Design*, 24

(10):531–540, 1992. URL http://dx.doi.org/10.1016/0010-4485(92)90040-h.

[62] Robert Joan-Arinyo and A Soto. A correct rule-based geometric constraint solver. *Computers & Graphics*, 21(5):599–609, 1997. URL http://dx.doi.org/10.1016/s0097-8493(97)00038-1.

[63] Bruno Buchberger. Gröbner bases: An algorithmic method in polynomial ideal theory, multidimensional systems theory, nk bose. *D. Reidel Publ. Co*, 1985. URL http://dx.doi.org/10.1007/978-94-009-5225-6_6.

[64] S Alasdair Buchanan and Alan de Pennington. Constraint definition system: a computer-algebra based approach to solving geometric-constraint problems. *Computer-Aided Design*, 25(12):741–750, 1993. URL http://dx.doi.org/10.1016/0010-4485(93)90101-s.

[65] Kunio Kondo. Algebraic method for manipulation of dimensional relationships in geometric models. *Computer-Aided Design*, 24(3):141–147, 1992. URL http://dx.doi.org/10.1016/0010-4485(92)90033-7.

[66] Alan Borning. The programming language aspects of Thinglab, a constraint-oriented simulation laboratory. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 3(4):353–387, 1981. URL http://dx.doi.org/10.1145/357146.357147.

[67] Christoph M Hoffmann and Pamela J Vermeer. A spatial constraint problem. In *Computational Kinematics' 95*, pages 83–92. Springer, 1995. URL http://dx.doi.org/10.1007/978-94-011-0333-6_9.

[68] Samy Ait-Aoudia, Brahim Hamid, Adel Moussaoui, and Toufik Saadi. Solving geometric constraints by a graph-constructive approach. In *Information Visualization, 1999. Proceedings. 1999 IEEE International Conference on*, pages 250–255. IEEE, 1999. URL http://dx.doi.org/10.1109/iv.1999.781567.

[69] William Bouma, Ioannis Fudos, Christoph Hoffmann, Jiazhen Cai, and Robert Paige. Geometric constraint solver. *Computer-Aided Design*, 27 (6):487–501, 1995. URL http://dx.doi.org/10.1016/0010-4485(94)00013-4.

[70] Samy Ait-Aoudia. *Modélisation géométrique par contraintes: quelques méthodes de résolution*. PhD thesis, Ecole Nationale Supérieure des Mines de Saint-Etienne, 1994.

[71] Christoph M Hoffmann, Andrew Lomonosov, and Meera Sitharam. Geometric constraint decomposition. In *Geometric constraint solving and applications*, pages 170–195. Springer, 1998. URL http://dx.doi.org/10.1007/978-3-642-58898-3_9.

[72] David Serrano. Automatic dimensioning in design for manufacturing. In *Proceedings of the first ACM symposium on Solid modeling foundations and CAD/CAM applications*, pages 379–386. ACM, 1991.

[73] László Lovász and Michael D Plummer. Matching theory. 1986. *Ann. Discrete Math*, 29, 1986.

[74] John E Hopcroft and Richard M Karp. An n^5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973. URL http://dx.doi.org/10.1109/swat.1971.1.

[75] Hichem Barki, Lincong Fang, Dominique Michelucci, and Sebti Foufou. Re-parameterization reduces irreducible geometric constraint systems. *Computer-Aided Design*, 70:182–192, 2016. URL http://dx.doi.org/10.1016/j.cad.2015.07.011.

[76] Hervé Lamure and Dominique Michelucci. Qualitative study of geometric constraints. In *Geometric Constraint Solving and Applications*, pages 234–258. Springer, 1998. URL http://dx.doi.org/10.1007/978-3-642-58898-3_12.

[77] Tiong-Seng Tay and Walter Whiteley. Generating isostatic frameworks. *Structural Topology*, (11), 1985. URL http://hdl.handle.net/2099/1047.

[78] Ruth Haas, David Orden, Günter Rote, Francisco Santos, Brigitte Servatius, Hermann Servatius, Diane Souvaine, Ileana Streinu, and Walter Whiteley. Planar minimally rigid graphs and pseudo-triangulations. In *Proceedings of the nineteenth annual symposium on Computational geometry*, pages 154–163. ACM, 2003. URL http://dx.doi.org/10.1016/j.comgeo.2004.07.003.

[79] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998. URL http://dx.doi.org/10.1145/272991.272995.

[80] Adel Moussaoui and Samy Ait-Aoudia. Some irreducible Laman graphs. Research report, Ecole Nationale Supérieure en Informatique, 22 june 2016. URL https://www.researchgate.net/publication/309718403_Some_irreducible_Laman_graphs.

[81] Robert Joan-Arinyo, Antoni Soto-Riera, Sebastià Vila-Marta, and Josep Vilaplana. On the domain of constructive geometric constraint solving techniques. In *Computer Graphics, Spring Conference on, 2001.*, pages 49–54. IEEE, 2001. URL http://dx.doi.org/10.1109/SCCG.2001.945336.

ملخص: يتكون النظام الهندسي المقيد من مجموعة محدودة من العناصر الهندسية، مثل النقاط، المستقيمات، الدوائر وغيرها، تربطها علاقات مختلفة كالبعد، الزاوية و التوازي. لهذه المسألة أهمية أساسية في العديد من التطبيقات، كالتصميم بمساعدة الحاسوب، النمذجة الهندسية ومؤخرا تحديد المكان في شبكات الاستشعار اللاسلكية. يتمثل حل النظام الهندسي المقيد في تعيين الإحداثيات الحقيقية للعناصر الهندسية في الفضاء الإقليدي. التقنيات التي تعتمد على المخطط البياني هي النهج السائد في الهندسة المستوية ولا سيما في سياق التصميم بمساعدة الحاسوب. لتسريع عملية الحل، عادة ما يتم تفكيك المخطط البياني الى مخططات جزئية، يتم بعد ذلك حل كل مخطط جزئي على حدى ليتم الحصول على الحل النهائي بواسطة جبر المخططات البيانية الفرعية. تركز معظم الأبحاث المنشورة و خاصة تلك التي تعتمد على المخطط البياني على التحليل النضري البحت مع بعض الأمثلة البسيطة. دائما ما تقارن الخوارزميات المقترحة مقارنة نظرية دون تقديم أي اختبارات على مخططات بيانية متعددة الخصائص الهيكلية مع درجات تعقيد مختلفة. المشكل يكمن في عدم توفر قاعدة بيانات لمسائل القيد الهندسي متدرجة التعقيد، و بقدر ما نعلم، لا توجد لحد الآن أي خوارزمية معروفة لإنشاء مخططات بيانية عشوائية ذات درجات تعقيد أو بنية هيكلية محددة. الهدف من هذه الرسالة هو تصميم مولد مخططات بيانية تستعمل لاختبار كفاءة أي خوارزمية في حل القيود الهندسية. لقد قمنا بتصميم مولد عشوائي بسيط، ولكنه فعال. يمكن استخدامه لجعل معايير الاختبارات اكثر اتساقا، أو مراقبة سلوك الخوارزميات مع قيود هندسية متباينة. يمكن لهذا المولد إنتاج مسائل مختلفة الأحجام والخصائص الهيكلية مع درجات تعقيد مختلفة . استندنا في تصميم المولد على تصنيفات المشكلة في الأعمال المنشورة سابقا. لقد أثبتنا أن المولد المقترح كامل، قابل للتخصيص، بسيط وفعال. تم التحقق من صحة ذلك بالتجربة و بالبرهان على بعض خصائصه نظريا.

Abstract: A geometric constraint system consists of a finite set of geometric elements, such as points, lines, and circles, along with relationships of different types such as distance, angle, incidence and parallelism. This problem is central to many applications, such as computer-aided design, molecular modelling and recently localization in wireless sensor networks. Solving a geometric constraint system consists of finding real coordinates of geometric elements in the Euclidean space. In 2-dimensional geometric constraint solving, graph-based techniques are a dominant approach, particularly in the computer-aided design context. To speed up the resolution process, these methods transform the geometric problem into a graph, which is decomposed into small subgraphs. Each one is solved, separately, and the final solution is obtained by recomposing the solved subgraphs. However, most of the previous research on graph-based approaches has only focused on the decomposition without any attention on what will be decomposed: the geometric constraint graph. Major proposed algorithms are discussed or compared theoretically, without presenting any tests on graphs instances with different structural properties, representing several cases of difficulties. Why? because as far as we know, there is no known algorithm for the creation of non-decomposable graphs or graphs with interesting structural properties that best highlight the efficiency of any algorithm. Our contribution is the design of a simple, but efficient random 2D geometric constraint graph generator. It can be used to make benchmarks for consistent tests, or to observe the behaviour of geometric constraints solving algorithms. It produces problem instances with various sizes and structural properties, covering different cases of complexity. Our design is based on the problem classification reported in the literature. We proved that our proposed generator is complete, customizable, simple and efficient. It has been validated experimentally and some of its properties have been theoretically proved.

Résumé : Un système de contraintes géométriques est constitué d'un ensemble fini d'éléments géométriques (points, lignes, cercles ...) et de relations géométriques tels que la distance, le parallélisme, l'angle, l'incidence, etc. Ce problème est central dans de nombreuses applications, en particulier la CAO. Résoudre un système de contraintes géométriques consiste à trouver des coordonnées réelles des éléments géométriques dans l'espace Euclidien de telle sorte que toutes les contraintes du système soient satisfaites. Dans l'espace 2D, les approches basées graphes sont dominantes. Afin d'accélérer la résolution, ces méthodes transforment le système de contraintes en un graphe qui sera ensuite analysé puis décomposé en petits sous-graphes. Chaque sous-graphe est résolu, séparément, la solution finale est obtenue en recomposant les sous-graphes résolus. Cependant, la plupart des algorithmes proposés sont analysés ou comparés théoriquement sans présenter des tests sur des cas de graphes de contraintes avec différentes propriétés structurales, représentant plusieurs cas de difficultés. Cela est dû à l'absence de méthodes connues pour la création de graphes de contraintes non décomposables ou ayant des propriétés structurelles qui illustrent mieux l'efficacité des algorithmes. Notre contribution est la conception d'un générateur aléatoire de graphe de contraintes qui est simple et efficace. Il peut être utilisé pour des analyses de performances, permettant des tests cohérents, ou bien pour observer le comportement des solveurs de systèmes de contraintes géométriques. Notre générateur produit des instances de problèmes avec différentes tailles et propriétés structurelles. Notre conception est basée sur la classification des problèmes rapportés dans la littérature. Nous avons prouvé que notre générateur est complet, paramétrable, simple et efficace. Il a été validé expérimentalement et certaines de ses propriétés ont été théoriquement prouvées.