



Ordre N°:

MOHAMED BOUDIAF UNIVERSITY- M'SILA
MATHEMATICS AND COMPUTING FACULTY
Computer Science Department

DISSERTATION

**Submitted in Partial Fulfillment of The Requirements
for The Degree of MASTER**

Field: Computer Sciences

Major: Information and Communication Technologies

By: Ridha DJAIDJA

THEME

**Rigidity Test Using the Pebble Game
Centralized Implementation**

Discussed publicly on: 31 may 2016 by the jury:

Mr. A. BOUGUERRA

University of M'sila

President

Mr. A. MOUSSAOUI

University of M'sila

Research Director

Mr. A. GUEMMOUGUI

University of M'sila

Member

June 2016

Dedication

I would like to dedicate this humble work to my beloved mother and my
generous father

To my lovely little sister, and all my venerable brothers

To my supervisor who really helped and guided me in a very tough times

A special dedication to all my teachers that thought me in my life since the
beginning

To all my dear friends and any person who helped and supported me during
conducting this work

Acknowledgement

I would like to thank my Supervisor Mr. Adel MOUSSAOUI, for his guidance. He has shown me what the role model of educator. Although he already has the experience for many years, he is always willing to learn new knowledge, and i am the beneficiary of the knowledge.

I thank him once more for the importance of theme he proposed.

I would thank the jury members for their patience in reading this dissertation.

Table of Contents

Dedication	ii
Acknowledgement	iii
Table of Contents	iv
Figure List	v
GENERAL INTRODUCTION	1
CHAPTER 1: RIGIDITY THEORY	4
1. Introduction	5
2. Bar and joint frameworks.....	5
3. Rigidity of Graphs in 2-dimensions	6
4. Rigidity of Graphs in 3-dimensions	11
CHAPTER 2: APPLICATIONS OF RIGIDITY	14
1. Introduction	15
2. Problem of Sensor Networks Localization	15
3. Rigidity and Flexibility of Protein Structures	19
3.1. Structural rigidity analysis of proteins	20
4. Application to computer aided design (CAD).....	23
CHAPTER 3: THE PEBBLE GAME ALGORITHM	26
1. Introduction	27
2. The pebble game algorithm	27
2.1. The begin of the pebble game algorithm.....	28
2.2. The end of the pebble game algorithm.....	29
2.3. The pebble game algorithm proceeds	31
CHAPTER 4: IMPLEMENTATION	33
1. introduction.....	34
2. Implementation.....	34
2.1. Part 1: Algorithm implementation	34
2.2. Part 2: visual representation	35
3. Experimental result	38
GENERAL CONCLUSION	41
References.....	43

Figure List

Figure 1.1 Bar and Joint Framework	5
Figure 1.2 (a) example of graph is flexible, (b) is minimally rigid.....	7
Figure 1.3 Using Laman’s Theorem for 2-dimensional graphs.....	9
Figure 1.4 A rigid graph with no triangles.	10
Figure 1.5 An example of the unsufficient of Laman type in 3-dimensions.....	12
Figure 2.1 (a) One possible localization. (b) Another localization.....	16
Figure 2.2 An example of a sensor network.....	17
Figure 2.3 A grounded graph, in which vertices a, b and c are beacons.....	18
Figure 2.4.....	19
Figure 2.5 Modelling molecular frameworks on the ethane molecule.....	21
Figure 2.6 Example of CAD System-AutoCAD	23
Figure 2.7 Decomposing a direction length design.....	24
Figure 2.7 Example Portion of Tower with Tubes	25
Figure 2.8 Rigidity Theory of Bridge in domain of Architecture.....	26
Figure 3.1 Algorithm for enlarging a pebble covering.	30
Figure 3.2 Example of The pebble game Algorithm proceeds.	32
Figure 4.1 (1) undirected representation of matrix, (2) example of a triangle.....	34
Figure 4.2 result of test for graph with four vertices	38
Figure 4.3 result of test for graph with six vertices.....	39
Figure 4.4 result of test for graph with six vertices.....	39
Figure 4.5 result of test for graph with twenty-six vertices	40

GENERAL INTRODUCTION

General Introduction

Rigidity theory has a rich history that can be traced back to Leonhard Euler, Baron Augustin Louis Cauchy, James Clerk Maxwell and a host of other mathematicians and engineers. Euler (1766) conjectured that "A closed spatial figure allows no changes, as long as it is not ripped apart" [7]. Maxwell (1864) was exploring whether structures were stable or unstable (deformable), and a host of other engineers worked on different applications, such as determining the structural stability of different truss configurations in bridges and in mechanical linkages.

Later in the half of the nineteenth century, Rigidity theory for plane bar and joint frameworks was already well advanced. Then earlier in twentieth century, previous work was summarized as well as important inductive techniques that are used to construct 2-dimensional bar and joint frameworks starting from the basic building blocks introduced by Henneberg [12]. These inductive techniques with other more recent techniques are still widely used, particularly in proofs and rigidity analysis of frameworks [27]. While a school of work in the rigidity theory continued for decades in the former Soviet Union [8], overall the best discoveries of the nineteenth century in the rigidity theory of bar and joint frameworks was forgotten for at least in the first half of the twentieth century.

An important advancement in rigidity theory begins with the theorem of Dutch mathematician Gerard Laman (1970), this theorem gives a beautiful and complete combinatorial characterization of rigidity and flexibility of bar and joint frameworks in dimension 2. In combinatorial characterization of rigidity, to determine the rigidity and flexibility of a corresponding framework it can simply count the vertices and edges in a graph and its subgraphs. Even though Maxwell and many others, with mechanical engineers, were counting vertices and edges in a graph to analyze rigidity, Laman's theorem gives a first rigorous characterization. Almost in similar time, mathematicians and engineers began to interact and further revive and develop the topic.

Mathematicians and researchers from different fields are continuing to work on problems and applications related to rigidity theory. The number of different applications that rely on this theory is rapidly expanding. Some of these include protein and glass network flexibility predictions [14, 24, 25], sensors and communication [1, 4], computer aided design (CAD), decomposition of linkages in mechanical engineering [20, 21] ... etc.

In our work we start by outlining some definitions from rigidity theory (Chapter 1). For the sake of simplicity, we will start by looking at the most common and simplest structures known as bar and joint frameworks. Since the fast algorithms for determining the rigidity/flexibility do not rely on the complicated geometry (positions). in other words, the rigidity becomes a graph theoretic property and this gives fast algorithms (pebble game) for determining rigidity/flexibility. We will also briefly present some problems with finding the fast algorithms for determining flexibility of the bar and joint frameworks in 3-dimensional space.

Chapter 2 presents a three different applications related to rigidity theory and we give some detailed examples, we will discuss for the 1- problem of network localization problem, 2- Rigidity and Flexibility of Protein Structures and 3-Application to computer aided design (CAD).

In Chapters 3 we will present some definitions and describes of the pebble game algorithm, also we illustrate example in detailed of the algorithm for enlarging a pebble covering.

Chapter 4 discussing for our implementation of pebble game algorithm and present some describe for the development environment, also our tools uses in implementation, and offer some concluding remarks.

CHAPTER 1
RIGIDITY THEORY

1. Introduction

Rigidity theory for bar and joint frameworks already well advanced. In this chapter, we present the basic concepts of rigidity theory, bar and joint frameworks and we introduce the rigidity of graphs in 2-dimensions and rigidity in 3-dimensions and the related discussions.

2. Bar and joint frameworks

This section present the most widely studied structure, which is composed of bars (rods) and joints, we will concentrate in 2-dimensional (plane) bar and joint frameworks, we notice that definitions and concepts in 2-dimensions have almost natural extensions and generalizations in 3-dimensional and higher dimensional space. bar and joint frameworks work as a good starting point in order to introduce the widely-used definitions from rigidity theory, and these structures are simple enough that we can visually perceive and appreciate most of the necessary concepts and definitions.[13]

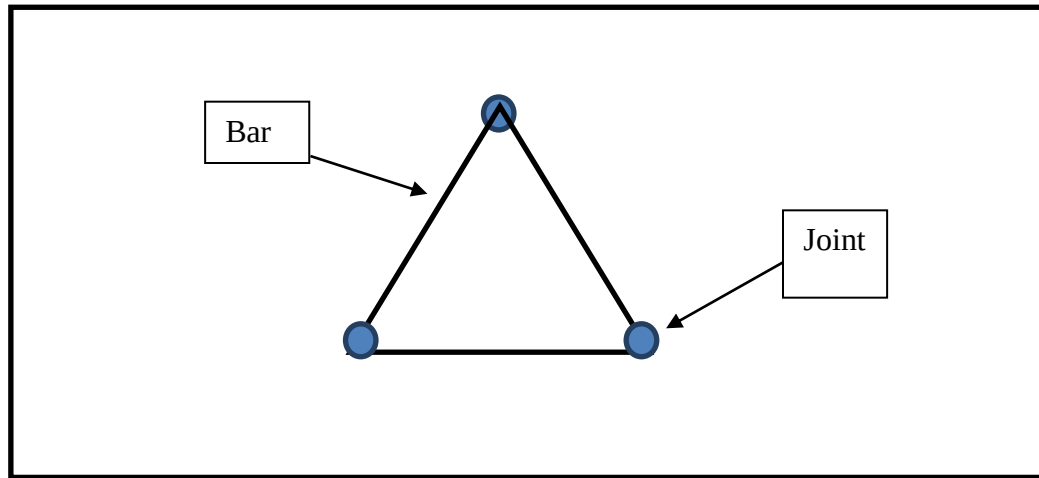


Figure 1.1: Bar and Joint Framework

In bar and joint frameworks, the bars which associate a couple of joints are supposed to be perfectly rigid and the joints (universal joints) are completely flexible (able to rotate) [10].

The joints or points in 2-dimensional space work as a connection between groups of bars, which only impose the restriction that the bars share the common endpoints. The bars correspond to fixed distances between some pairs of joints. If we move such a framework in the plane, consisting of bars linked to their ends, and the distances between all pairs of joints that are

connected by a bar will stay fixed during the movement. Then we should ask this interesting question: will the distances between other pairs of joints also stay fixed? [13]

Here we are concerned about understanding the movement of framework, movements will let us to get answers about rigidity/flexibility of this framework.

Deformation: is a movement that keeps the lengths of all bars of the framework but changes the distance between some pairs of (unconnected) joints of the framework. [13]

If no deformation exists, then the movement is called *rigid motion*. A rigid motion of a framework preserves the distance between all joints in the framework whether joints are connected by a bar or not. As a result, rigid framework is a framework with no deformations, and flexible framework is a framework with deformations.

We are interested in motions of a framework that go beyond the ever-present trivial motions, that is motions other than those from congruence for example in translations and rotations. [13]

3. Rigidity of Graphs in 2-dimensions

We can completely ignore the location of the vertices and the precise distances of the edges that are present in the framework (Gluck) [4] and in its place, we focus on the topological properties of the framework. In a different words, to understand the rigidity/flexibility properties of the framework, we concentrate on the underlying graph $G=(V,E)$: rigidity becomes a combinatorial property. In this spirit, we can now step back from our discussion about rigidity/flexibility of frameworks and start talking about the rigidity/flexibility of graphs. [13]

When we talk about the rigidity/flexibility of graphs the following question came in mind: how can we recognize if the graph in 2-dimensions is rigid? What follows is an argument that will allow us to predict the answer. Most of this already follows in one form or another from Theorem 1.3.1, this will prove to be central when we look at the algorithmic approach for determining the graph rigidity/flexibility. Moreover, this matches more closely the vocabulary in graph rigidity and in algorithmic-rigidity literature.

Because edges are constraining the possible movements of vertices, it is clear that a graph with many edges have probability to be rigid more than a graph with only few edges. We need keep in mind, that no matter how many edges the graph has, we will never constrain the three trivial degrees of freedom (motions of a rigid body). We say that a graph is minimally rigid if it is rigid and removing any one of its edges (but not vertices) makes the graph non-rigid. So, the ultimate question is what is the condition for the graph to be minimally rigid?

Flexibility in the graph occurs due to unconstrained degrees of freedom. Recall, in 2 dimensions a single vertex has two degrees of freedom, so it is clear that a graph which has n vertices ($n = |V|$) has at most $2n$ degrees of freedom (it has exactly $2n$ degrees of freedom when no edges are present). Each edge eliminates at most a single degree of freedom, so in order to eliminate all the internal degrees of freedom, then one could anticipate that at least $2n-3$ edges are necessary for rigidity in a graph. However, having the required total number of edges in the graph is clearly not sufficient, as the edges could be crowded between only a select few vertices, leaving the other vertices unconstrained (see Figure 1.2 (a)). The goal then, is to have $2n-3$ well-distributed edges. That is, in graphs with $2n-3$ edges, no subgraph on n' vertices should have more than its fair share of $2n'-3$ edges (see Figure 1.2 (b)). This criterion is sometimes called the Laman condition, due to Laman [11], and the graphs (subgraphs) that satisfy this condition are known as Laman graphs (subgraphs). If any subgraph $G(V')$, where $|V'| = n'$, has more than $2n'-3$ edges, then some edges are redundant. Non-redundant edges are independent. We say that a graph is stressed if it has at least one redundant edge. Removing a redundant edge from the framework does not affect the rigidity of that framework (degrees of freedom of the graph before and after its removal stay the same). [13]

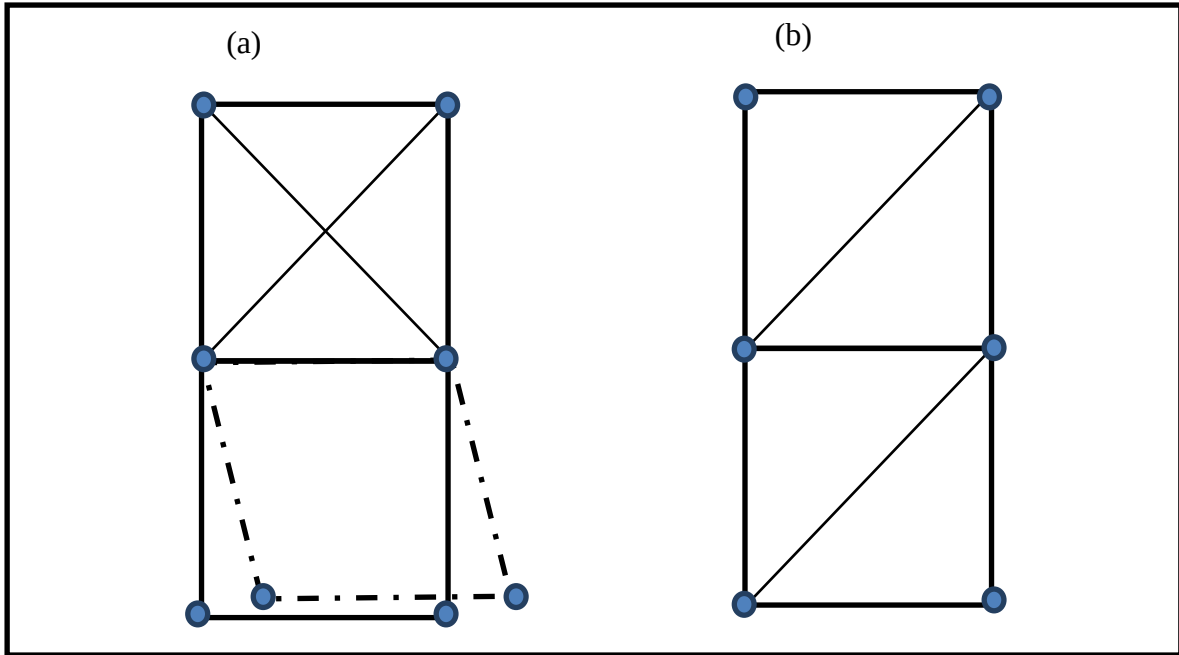


Figure 1.2 (a) example of graph is flexible, (b) is minimally rigid

In figure 1.1, both graphs have $2(6) - 3 = 9$ edges. but because the edges in graph (a) are not well-distributed (the subgraph induced by the top four vertices has more edges than required, $2(4)$

- 3 = 5 edges are required, but it has 6 edges, which means that one edge is redundant), so this graph is flexible. On the other hand, the graph in (b) is minimally rigid as all the edges are well distributed. . [13]

This basic intuition is correct in 2-dimensions and it is confirmed in the next well-known theorem due to Laman (1970) [18]:

Theorem 1.3.1(Laman) The edges of a graph $G = (V, E)$ are independent in 2-dimensions if and only if no subgraph $G' = (V', E')$ has more than $2n'-3$ edges ($n' > 1$), where $n' = |V'|$.

Corollary 1.3.2 A graph $G = (V, E)$ with $2n-3$ edges ($n > 1$) is minimally rigid in 2-dimensions if and only if no subgraph $G' = (V', E')$ has more than $2n'-3$ edges, ($n' > 1$), where $n' = |V'|$.

Laman's Theorem is the key graph theoretical result in the plane that allows us to characterize minimally rigid graphs. Several other equivalent characterizations have since been discovered [11], and some of these play crucial roles in finding polynomial time algorithms for testing rigidity of graphs (see below).

Laman's Theorem allows us to test whether the graph is rigid or flexible (generically) by simply *counting* the edges and their distribution in the graph (combinatorial property).

As a historical note, we should point out that even though Laman was the first to prove this result, this requirement had been stated in the nineteenth century. [13]

The Figure 1.3 gives few examples that show the usefulness of the Laman's Theorem.

- The graph (a) is minimally rigid.
- The graph (b) is rigid and stressed.
- The graph (c) is flexible and stressed.
- The graph (d) is flexible (it has less than eleven edges).

Consider the graphs in Figure 1.3: All of the graphs have seven vertices, so in order for the graph to be minimally rigid we need a total of eleven ($2(7) - 3 = 11$) edges and all of the edges should be well distributed.

→The graph (a) has eleven edges and by inspection we can see that no subgraph of n' vertices has more than $2n' - 3$ edges, so, all eleven edges are independent (well distributed). Therefore, by Laman's Theorem this graph is minimally rigid.

→The graph (b) is rigid, but it is not minimally rigid (we added an extra edge) it has 12 edges. This graph is also stressed. It is clear that any graph that is rigid and not minimally rigid is also stressed (there is a presence of redundant edge), also known as an over-constrained graph.

→ The graph (c) is flexible. This graph has eleven edges, but one subgraph (indicated by red edges and its end vertices) does not satisfy the Laman count. The edges in this subgraph are redundant. This graph shows that a flexible graph can be stressed. Note that redundant edges always belong to some rigid and stressed subgraph.

→ Finally, the graph (d) is flexible graph, as it needs another edge.

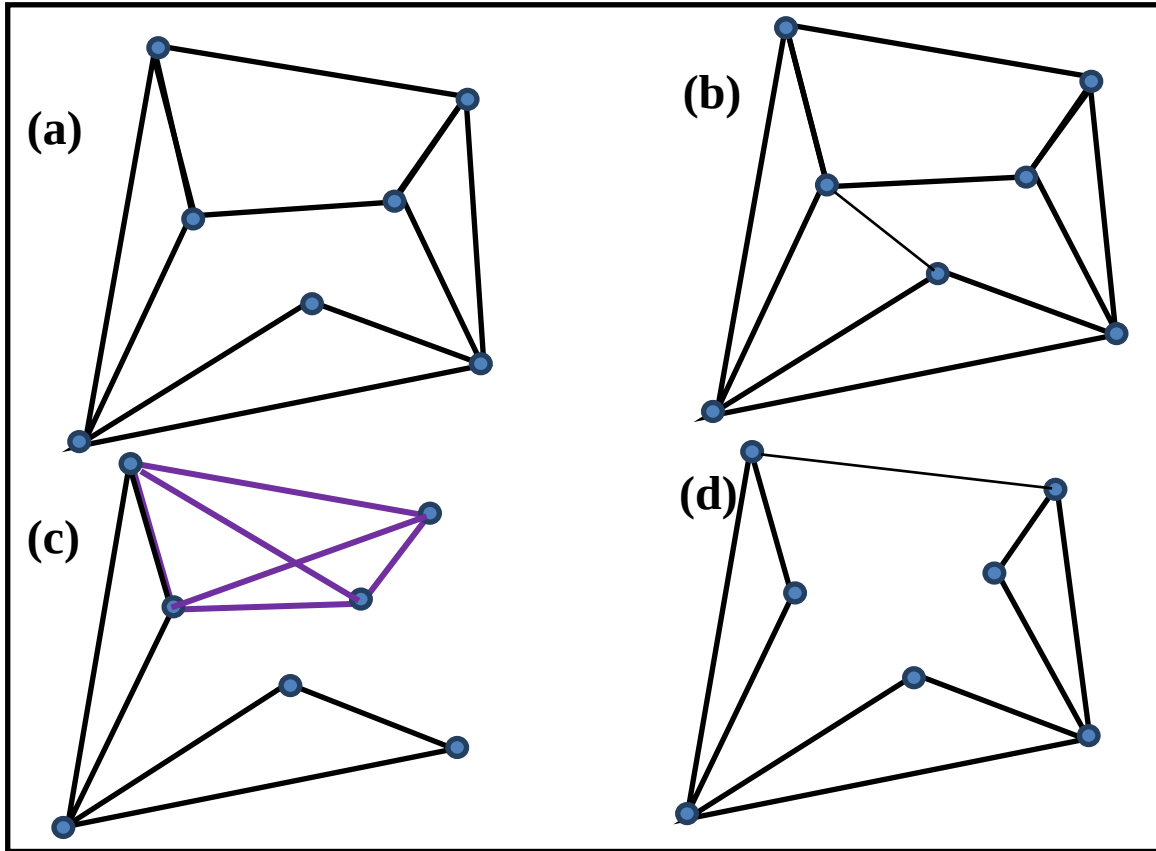


Figure 1.3 Using Laman's Theorem for 2-dimensional graphs.

This examples illustrates that flexibility in the graph (in 2-dimensions) occurs because of the edges are not well distributed, or there are too few edges in the graph.

Note that triangles are not required for rigidity. There are many graphs that do not have triangles, and yet they are rigid. One example is the graph of the complete bipartite graph $K_{3,3}$ in Figure 1.4. It has $2(6) - 3 = 9$ edges, and as no subgraphs have too many edges, all edges are independent (well-distributed). [13]

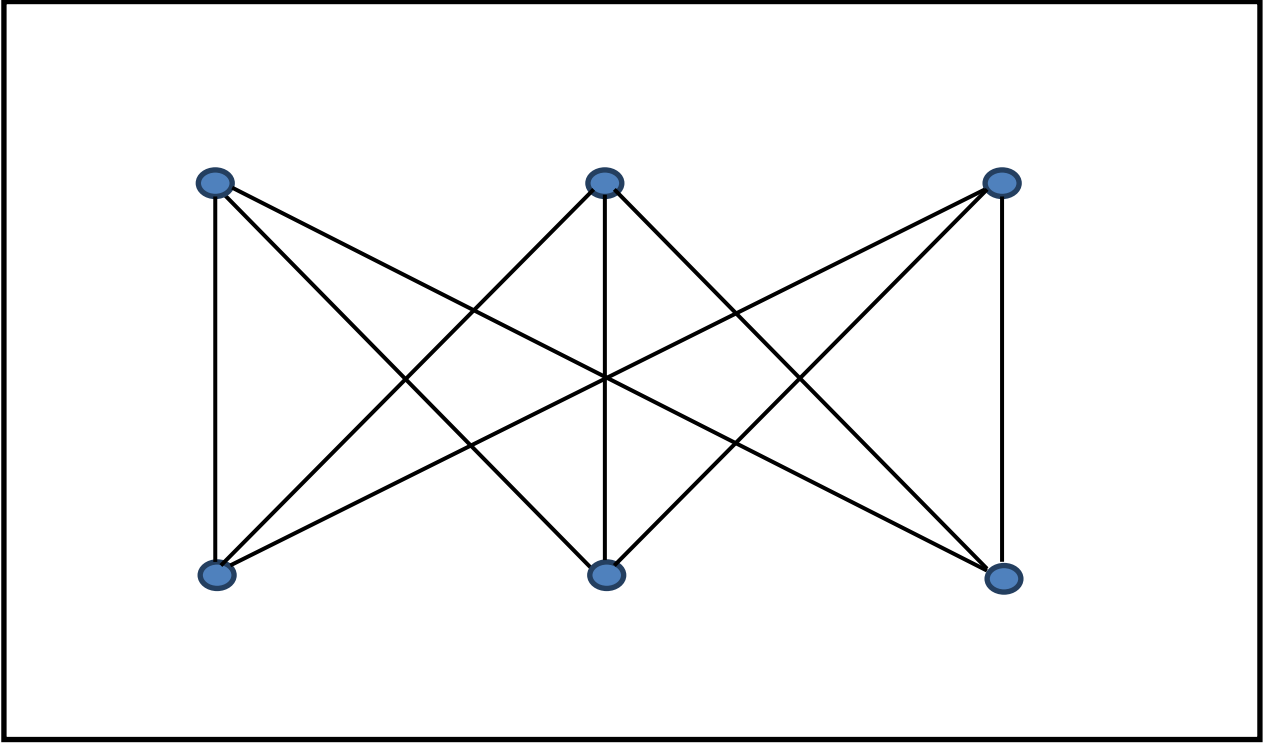


Figure 1.4: a rigid graph with no triangles.

Laman's Theorem is an elegant and remarkable result, but in its original form, it is obvious that it gives a poor algorithm. It requires counting the number of edges in every subgraph, of which there are an exponential number [17]. We would like to have a fast and efficient algorithm that we can check this count for rigidity.

Laman's Theorem has several equivalent formulations. One particular formulation, originally due to Hendrickson [11] is:

Theorem 1.3.3 For a graph $G = (V, E)$ having m edges and n vertices, the following are equivalent.

- (i) The edges of G are independent in 2-dimensions.
- (ii) For each edge $\{i, j\}$ in G , the graph formed by adding three additional edges $\{i, j\}$ has no n' vertex subgraph with more than $2n'$ edges.

Hendrickson (as a bipartite matching algorithm) initially used this formulation. Then later it revised by Jacobs and Hendrickson [17], and developed into a particularly intuitive algorithm, the pebble game algorithm. One of the most distinctive features of the pebble game algorithm is that it is computationally efficient.

We are going to give the details of the 2-dimensional pebble game algorithm; it is well known that the main process of all the pebble game algorithms is essentially the same, with some minor modifications. This description of the pebble game algorithm will be seen below.

The idea behind the pebble game algorithm is to grow a maximal set of independent edges one at a time by matching them to the degrees of freedom in the graph (pebbles). A new edge is added if it is determined to be independent of the existing set. So, for 2-dimensional graphs (bar and joint frameworks), if $2n-3$ independent edges are found (n is the number of vertices in the graph) then the graph is rigid, the pebble game gives a unique and visually appealing way to test edges for independence, and outputs several key rigidity properties. . [13]

4. Rigidity of Graphs in 3-dimensions

The graph theoretic analysis of rigidity of frameworks in 3-dimensions is complicated, not fully understood phenomenon. The equivalent restatement of the Laman condition for (minimal) rigidity of 3-dimensional graphs is:

“A graph $G = (V, E)$ is minimally rigid in 3-dimensions ($n > 2$) if and only if it has $3n-6$ edges and no subgraph $G' = (V', E')$ has more than $3n'-6$ edges, where $n' = |V'|$ ”.

This means that we need enough edges for the graph to be rigid, and all edges should be well distributed so that there is no packing of too many edges between a given set of vertices. Unfortunately, for all of the effective results that the Laman condition provides in 2-dimensions, the $3|V|-6$ count is not sufficient in 3-dimensions.

The graph of the "double banana" in Figure 1.5 it has the exact required number of edges and there are no subgraphs having more than $3n'-6$ edges connecting n' vertices. Accordingly, all eighteen edges are "well-distributed", but this graph is flexible. Each banana (a rigid subgraph) is made of two tetrahedra glued together along a triangle, and the two bananas are attached at their tips (top and bottom vertices). It is clear that the two bananas can twist independently around the axis through their tips. This imaginary axis called the 'implied hinge' (two rigid subgraphs of the graph rotate around the hinge). . [13]

It is conjectured that the only reason the Laman's counts fail in 3-dimension, is because of the presence of implied hinges. This is known as the Dress conjecture. While the double banana is the smallest example where Laman's condition is insufficient, several other ones have also been classified [16].

The problem with the "double banana" example can be seen in Figure 1.5 (the double banana, has 18 well-distributed edges, but it is still flexible).

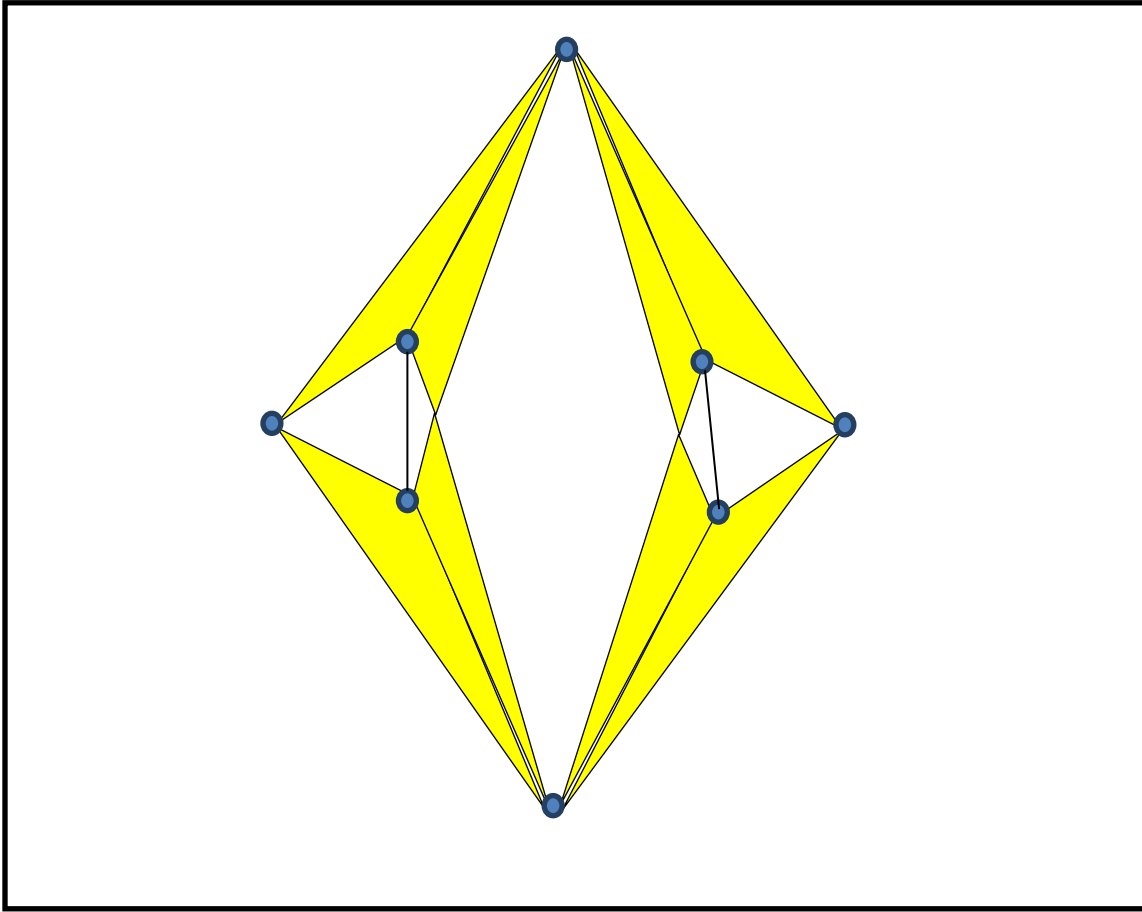


Figure 1.5 An example of the insufficiency of Laman type in 3-dimensions.

The problem with the Laman type in 3-dimensions is that it does not work for a single edge (i.e. when $n = 2$) [26]. Not only does the Laman type of count fail here, but also there are currently no known polynomial time algorithms for testing rigidity for general 3-dimensional graphs [26]. A lot of work and extensive research has been done on this problem and yet only some partial results can be traced. . [13]

Rigidity in 3-dimensional graphs stayed one of the big open problems in rigidity theory.

CHAPTER 2

APPLICATIONS OF RIGIDITY

1. Introduction

Although rigidity has been studied since the time of Lagrange (1788) and Maxwell (1864), it has begun to find applications in the basic sciences. The modern era starts with Laman (1970), who made the subject rigorous in two dimensions, followed by the development of computer algorithms that can test over a million sites in seconds and find the rigid regions, and the associated pivots, leading to many applications. In this chapter we describe some of these applications:

2. Problem of Sensor Networks Localization

In [3], Aspnes et al. provided a theoretical foundation for the problem of network localization problem in which some nodes know their locations and other nodes determine their locations by measuring the distances to their neighbors. More precisely, one begins with a network N in real d -dimensional space (where $d = 2$ or 3) consisting of a set of $m > 0$ nodes labeled 1 through m that represent the special "beacon" nodes together. Each node is located at a fixed position in $\mathcal{R}^d$ and has associated with it a specific set of "neighboring" nodes. It is required that the definition of a neighbor is a symmetric relation on $\{1, 2, \dots, n\}$ in the sense that node j is a neighbor of node i if and only if node i is also a neighbor of node j . Under these conditions, N 's neighbor relationships can be described by an undirected graph $G_N = (V, E_N)$ with vertex set $V = \{1, 2, \dots, n\}$ and edge set E_N defined so that (i, j) is one of the graph's edges precisely when nodes i and j are neighbors.

The network localization problem with distance information is to determine the locations of all nodes p_i in $\mathcal{R}^d$ given the graph G_N of the network, the positions of the beacons $p_j, j \in \{1, 2, \dots, m\}$ in $\mathcal{R}^d$, and the distance $\delta_N(i, j)$ between each neighbor pair $(i, j) \in E_N$. We say that a network localization problem is solvable if there exists a unique set of nodes $\{p_{m+1}, p_{m+2}, \dots, p_n\}$ in $\mathcal{R}^d$ consistent with the given data $G_N, \{p_1, p_2, \dots, p_m\}$, and $\delta_N: E_N \rightarrow \mathcal{R}$. The graph $G_N = (V, E_N)$ is unique localizable if its corresponding network localization problem is solvable [30].

Before going further, we give an example of a graph which is not unique localizable. Consider the graph in Figure 2.1. Suppose that vertices a, b and d know their locations. Hence the distances between a and b , b and d , and also a and d are known. Suppose vertex c knows only its distances to a and b , but not the distance to d . Then there are two possible

locations for c as shown in Figure 2.1 (a) and (b). Consequently, the graph in Figure 2.1 is not unique localizable [30].

A sensor network consists of multiple detection stations called sensor nodes, each of

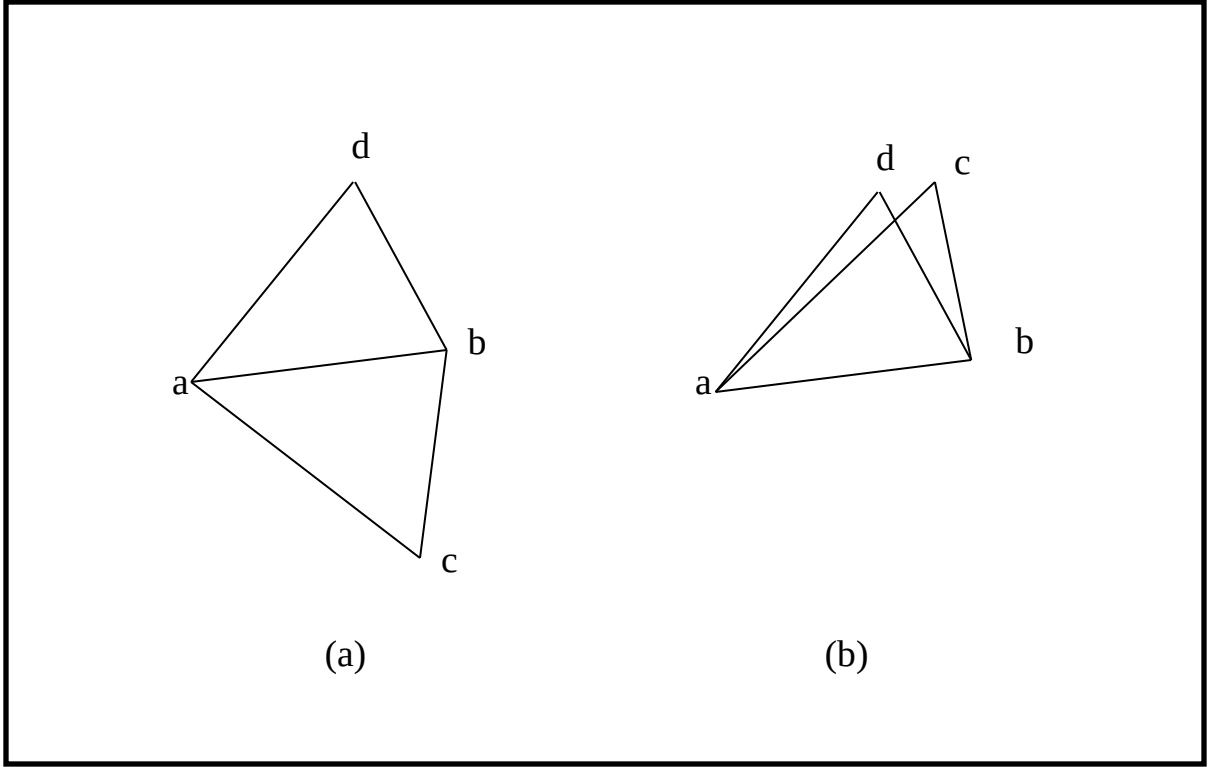


Figure 2.1 (a) One possible localization. (b) Another localization.

them is small, lightweight and portable. Every sensor node is equipped with a transducer, microcomputer, transceiver and power source. The transducer generates electrical signals based on sensed physical effects and phenomena. The microcomputer processes and stores the sensor output. The transceiver, which can be hard-wired or wireless, receives commands from a central computer and transmits data to that computer. The power for each sensor node is derived from the electric utility or from a battery. Figure 2.2 is an example of a sensor network [30].

A sensor network can be used to monitor and record conditions at diverse locations. The monitored parameters may be temperature, pressure, humidity, illumination intensity, wind direction and speed, vibration intensity, sound intensity, chemical concentrations, pollutant levels and vital body functions. In a sensor network, every sensor needs to know its location to detect and record events and to route packets [26]. So the network localization problem plays an

important role for sensor networks. Recently, several methods [9, 17, 18] have been proposed to determine the location of the sensors in a sensor network [30].

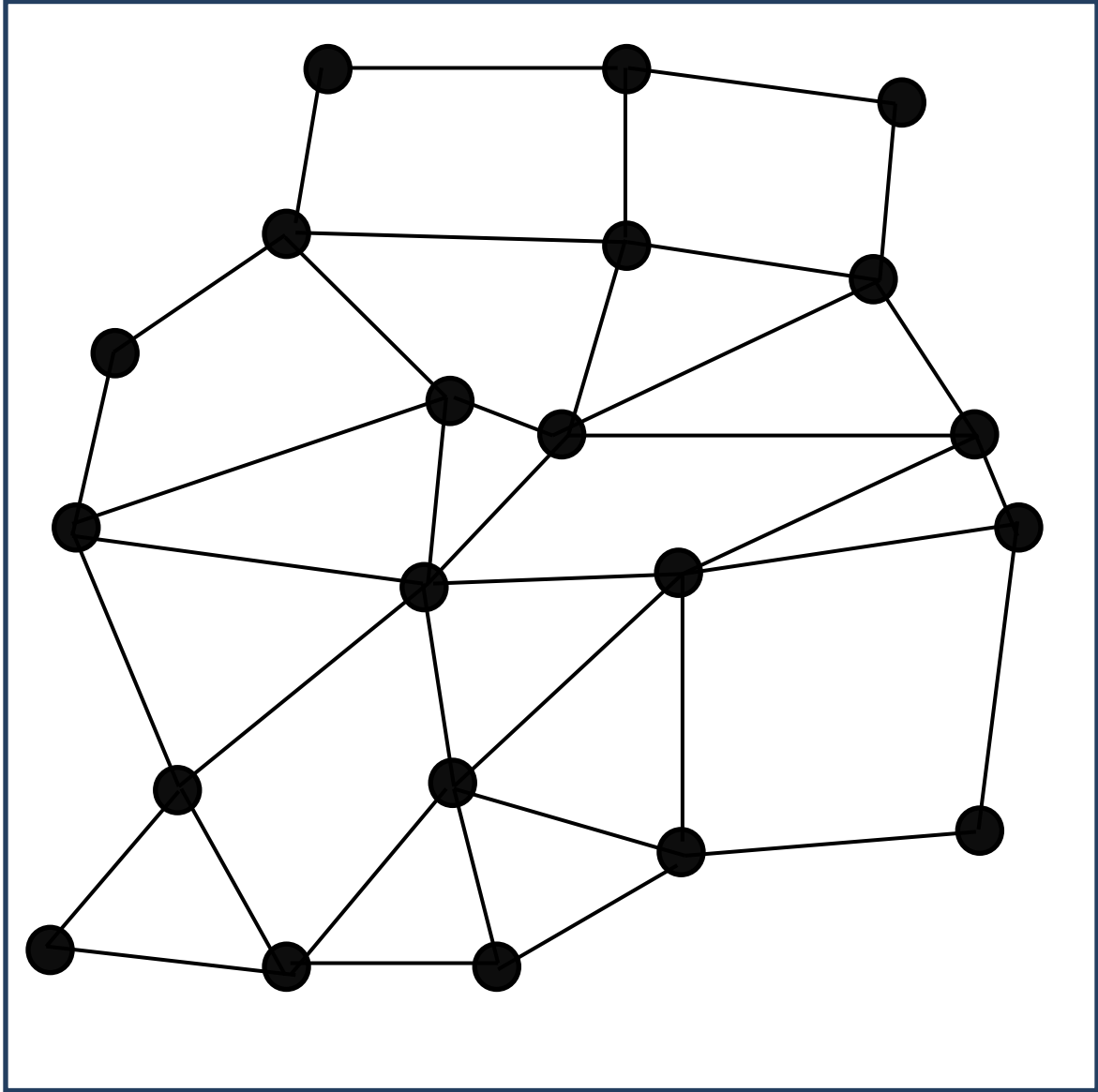


Figure 2.2 An example of a sensor network.

a node and a vertex are used interchangeably. A graph G is k -connected if it remains connected upon the removal of any set of $< k$ vertices. The connectivity of a complete graph of order n is defined to be $n - 1$ [30].

In [3], Aspnes et al. proposed the idea of grounded graphs. More precisely, in a grounded graph, each vertex represents a node in the given network and there is an edge between the two vertices if the distance between the two nodes is known. It has been assumed that when a sensor

network is given, the locations of beacons are known. Hence the distance between any two beacons is implicitly known. Consequently, in a grounded graph, any two beacons are connected and therefore the subgraph graph induced by the beacons is a complete graph. In order to determine the locations, other sensor nodes must compute the distances between them and the beacons or the distances between them and those sensor nodes which have already known their locations. As an example, in Figure 2.3, the vertices a , b and c are beacons and distances between vertices a and d , b and d , c and d , a and e , c and e , d and e are known. Because the vertices a , b and c are beacons, the distance between each pair of them is implicitly known. The vertex d can compute the distances between it self and the vertices a , b and c to determine its location. After d knows its location, the vertex e can compute the distances between it self and the vertices a , c and d to determine the location of e [30].

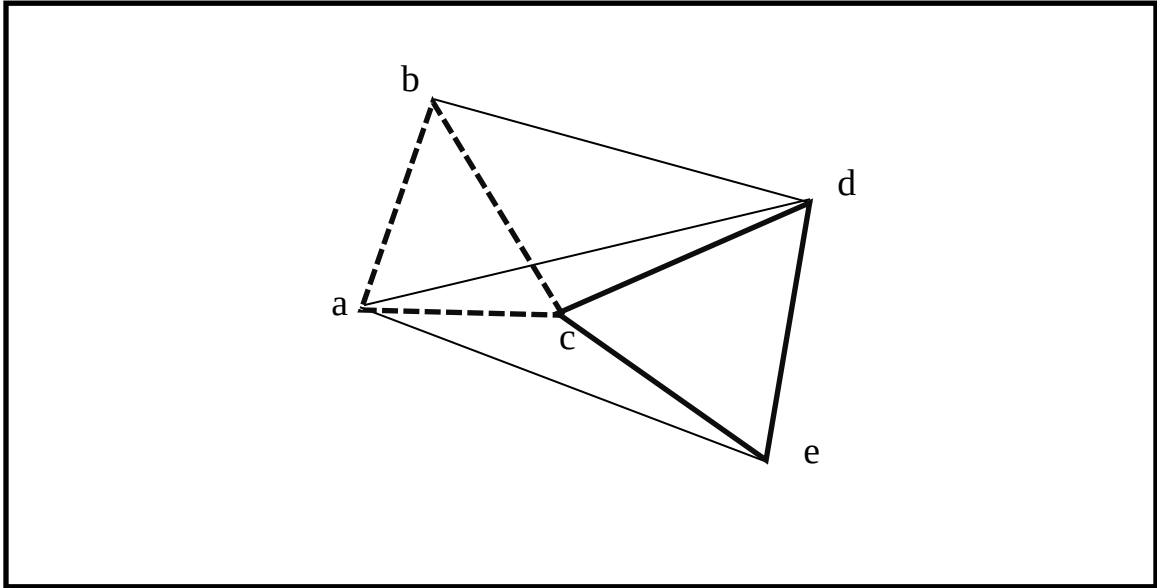


Figure 2.3 A grounded graph, in which vertices a , b and c are beacons.

It has been proven in [3] that: A network has an unique localization if and only if its corresponding grounded graph is globally rigid. Formal definitions of the rigidity and globally rigidity of a graph is quite copious and can be found in [3]. In [15], it has been proven that a graph of order n must have at least $2n - 3$ edges in order to be rigid. In [15], Laman proposed the following theorem for determining if a graph with $2n - 3$ edges is rigid; for convenience, call this theorem Laman's Theorem [30].

Laman's Theorem a Graph G of order n and size $2n - 3$ is rigid if and only if it has no subgraph with more than $2n' - 3$ edges, where n' is the number of vertices in the subgraph.

A graph is redundantly rigid if it remains rigid after removing any single edge. It has been proven in [35] that a graph G with ≥ 4 vertices is globally rigid in $\mathbb{R}^2$ if and only if it is 3-connected and redundantly rigid in $\mathbb{R}^2$. As an example, consider Figure 2.4. The graph in Figure 2.4(a) is not rigid. The graph in Figure 2.4(b) is rigid but not redundantly rigid. The graph in Figure 2.4(c) is 3-connected and redundantly rigid, hence this graph is globally rigid [30].

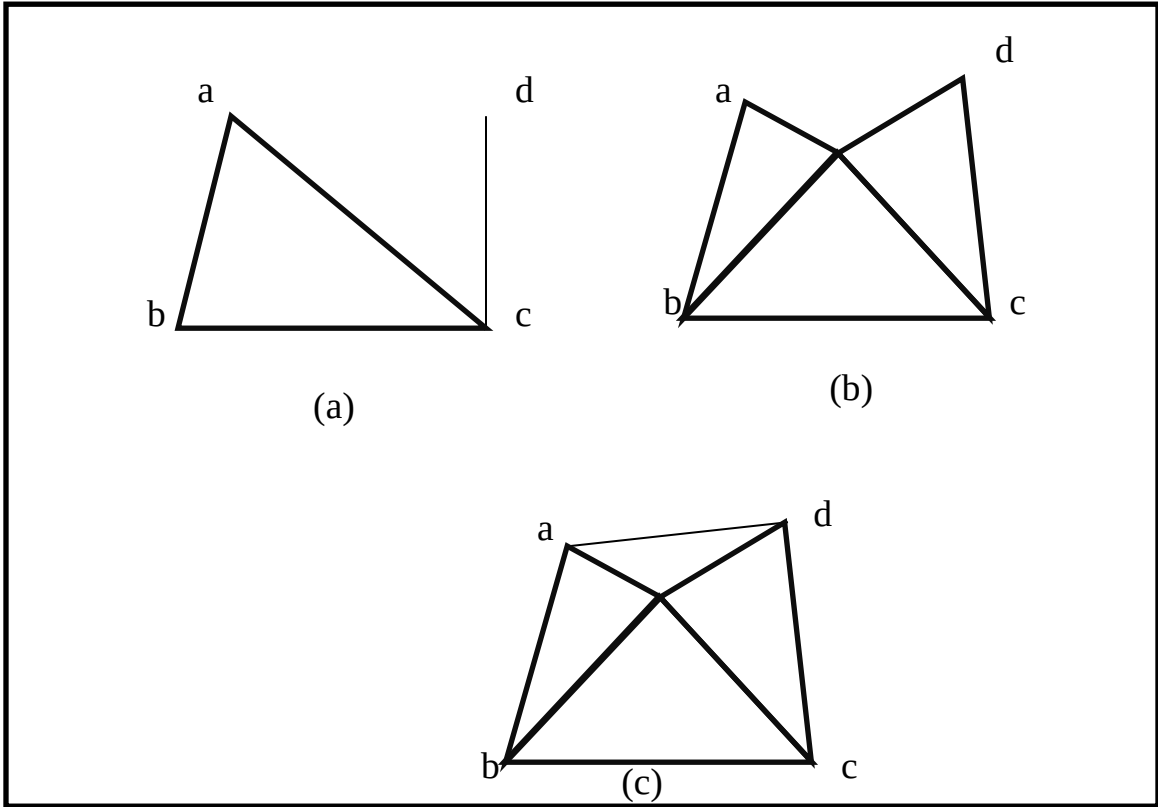


Figure 2.4 (a) A graph which is not rigid. (b) A graph which is rigid but not redundantly rigid. (c) A graph which is globally rigid.

3. Rigidity and Flexibility of Protein Structures

Rigidity and flexibility of proteins at the right place play an important role in protein function [5]. To describe protein rigidity properties mathematical structural rigidity theory is a useful tool, since proteins are real molecular frameworks. However, interactions between atoms cannot be considered as simple geometrical constraints, since they may fluctuate. Thus rigidity analysis of protein structures provides information on the phase space of collective motions of protein

segments. However, local flexibility/mobility properties of protein structures at the atomic level are also very important, and require other methods of protein dynamics for their study not reviewed here [29].

3.1. Structural rigidity analysis of proteins

Computer programs are available for the rigid cluster decomposition of proteins using three dimensional protein structures. Figure 2.5 illustrates the two possible major ways of modelling molecular frameworks on the ethane molecule. The first method uses the bar-joint model, in which, each atom is represented by a joint, and distance constraints of bonds are represented by bars. For modelling angle constraints of rotating bonds, such as single covalent bonds, additional bars connecting next nearest neighbors have to be added to the model. Torsional constraints of double covalent bonds can be fixed by introducing third-nearest distance constraints. The resulting network is the so called bond-bending network used and analyzed by MSU-FIRST software [13, 29].

In Figure 2.5(A) Molecular framework models are illustrated with the help of the ethane molecule having a rotating single covalent bond between carbon atoms. And at Figure 2.5 (B) In the bond-bending network nodes represent atoms, solid lines represent distance constraints belongs to central forces, dashed lines represent additional distance constraints to model angle constraints of rotating bonds. Figure 2.5 (C) In the bodybar-hinge representation rigid bodies are set of atoms rigidly attached to each other's. Rotational degree of freedom of a single covalent bond is modelled as a hinge, shown by dashed (white) line in the figure [29].

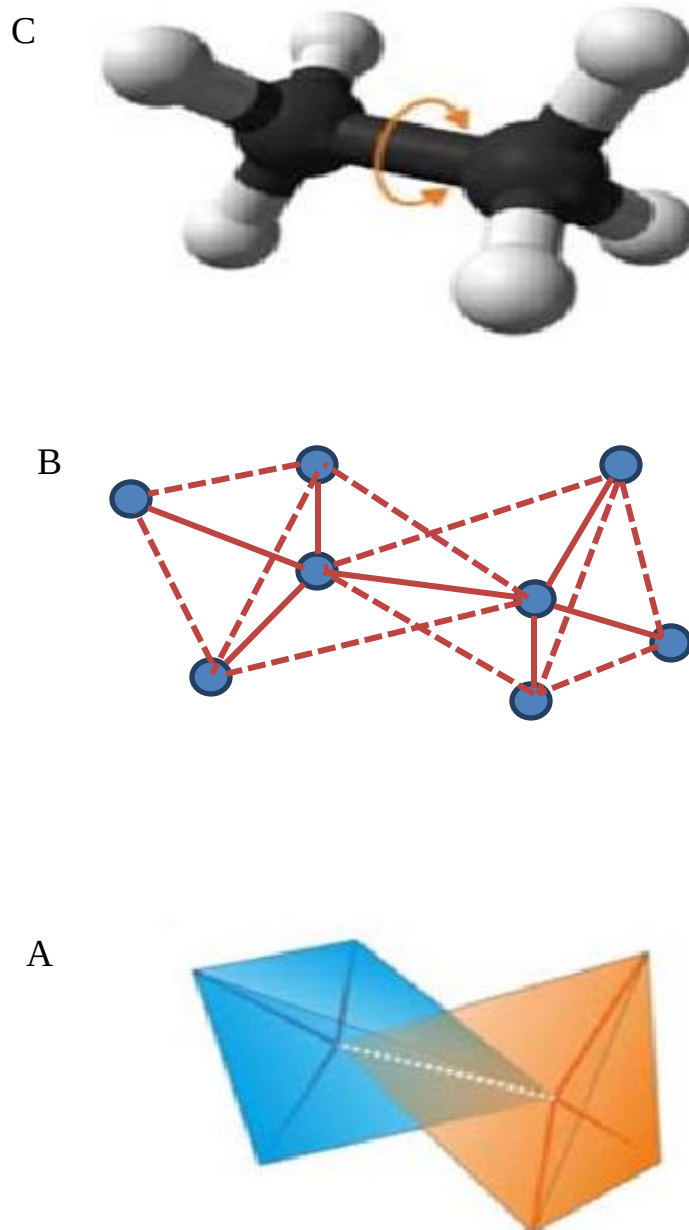


Figure 2.5 Modelling molecular frameworks on the ethane molecule

In the body-bar-hinge model, local rigid groups of atoms are represented by rigid bodies. For example, in the ethane molecule (C_2H_6) both carbons with their four bonds form a rigid tetrahedron (see panel C of Figure 2.5), and the rotating single covalent bond between the carbons is modelled by a hinge. Double covalent bond between carbons does not allow any rotational freedom, therefore an ethylene molecule (C_2H_4) itself is a rigid body. Weaker interactions can be modelled by specifying a number of bars (from 1 to 6), each one removing one degree of freedom. The body-bar-hinge model is implemented in by the KINARI web server [6]. The default model of KINARI uses hinges for modelling single covalent-, disulphide- and strong hydrogen bonds, six bars for double covalent- and resonant bonds, and two bars for hydrophobic interactions [29].

KINARI has the great advantage, that it has an on-line visualization plug-in for studying the results of the rigidity analysis [19]. Both FIRST and KINARI give the rigid cluster decomposition of the protein and a number of calculated rigidity measures, such as the widely used rigidity order parameter meaning the relative size of the largest rigid cluster as compared to the whole structure. It is also possible to classify each interaction in terms of its contribution to the overall rigidity using redundancy analysis [6, 29].

Rigid cluster decomposition methods have also been applied for other macromolecules, such as for RNA. Since molecular dynamics calculations producing individual pathways in protein folding are computationally very expensive, the much faster rigid cluster decomposition methods are used to reduce the conformation space efficiently and improve dynamical simulations [23, 29].

Rigid cluster decomposition is a very useful tool, but it is important to note its limitations.

- In both molecular framework models, we have to decide, which bonds and interactions are treated as a strict and exact integer number of constraints, and all other forces are completely ignored.
- Generic rigidity depends on the network topology and not on the geometry. In reality the possibility that the framework is not generic is close to zero, but there are always lots of nodes near degenerate position (for example arranged in a straight line), which can be important, since real chemical bonds have dispersion in length and dihedral angles[29].

The above arguments suggest that for real structures rigidity can be characterized more likely with the help of probabilistic quantities, rather than by definite cluster decomposition. The

recently developed generalization by Gonzalez et al. [11] offers a very helpful solution to this problem. This generalization is called as "virtual pebble game", and represents a non-integer probabilistic constraint counting algorithm running on weighted interaction networks [29].

4. Application to computer aided design (CAD)

The Computers-Aided Design is use of computer to assist for creation and manipulation- design model represented as a digital data or computer image to assist engineering the design process. a CAD system usually consists of:

1. Hardware: the computer and associated peripheral equipment.
2. Software: the computer program(s) running on the hardware.
3. Data or model created and manipulated by the software.
4. Human knowledge and activities.

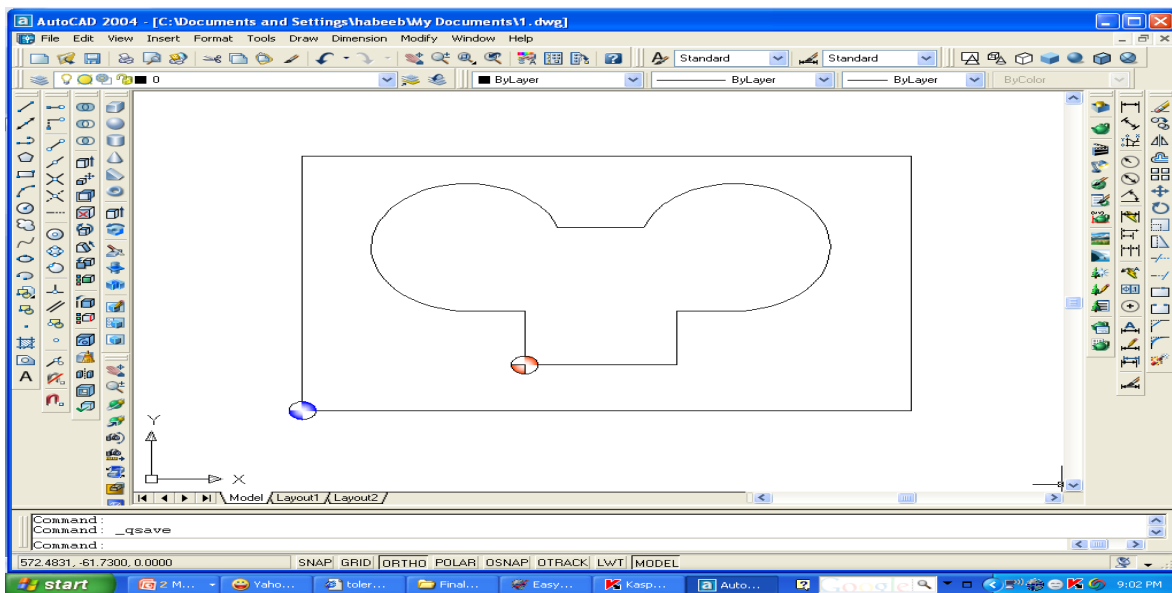


Figure 2.6 Example of CAD System-AutoCAD

In computer aided design (CAD), the object is to specify and represent a complicated design using a standard collection of geometric objects such as points, circular arcs, line segments, etc. The design is specified by requiring the objects to satisfy certain constraints. For example, certain point-line incidences, segment lengths, or angles may be prescribed. The basic design problems are: realizability, (do the constraints determine the congruence class of the design?), and constraint independence - (are all the constraints necessary?) [31].

Theoretically, just as with rigidity of frameworks, the constraints may be written as a system of algebraic equations whose variables are the coordinates or other parameters of the geometric objects, see [22]. The rank of the Jacobian of the system of constraints is used to answer some of the basic design questions. Even these linear computations may be slow or unstable because of degeneracies. Since typical CAD designs use hundreds of objects, avoiding degeneracies to achieve computational stability is useful, and may be accomplished by using generic parameters [31].

For plane length designs, where the objects are points and the constraints specify distances between certain pairs of points, rigidity theory provides a purely combinatorial solution to generic design questions. Parallel drawings serve the same role for direction designs. For designs with both direction and length constraints, see [19], an analogue of Laman's theorem is also valid [31].

Purely combinatorial characterizations are available in this setting as well: A set of direction and length constraints is independent and achieves local uniqueness if and only if there is a decomposition of the edges of the underlying graph into two spanning trees, such that no two proper subtrees of the same constraint type have the same span. Figure 2.6 shows a two tree decomposition of a direction length design in which the edges which correspond to direction constraints are indicated by arrows in their interior [31].

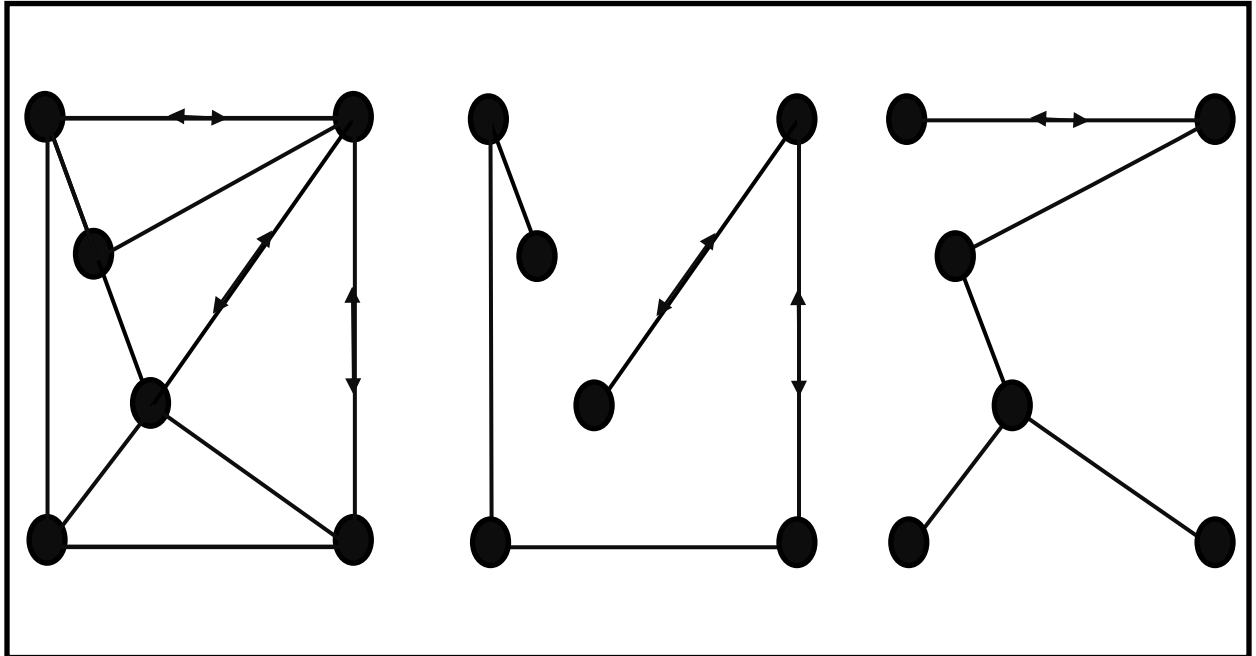


Figure 2.7 Decomposing a direction length design.

Other combinations of CAD motivated constraint systems have not yet been satisfactorily analyzed. Observe that a direction constraint corresponds to an equation among the coordinates of two points, and the same is true for a length constraint. An angle constraint, by contrast, involves three points. A theory of designs mixing lengths and angles would be very interesting for map making however, so far, none has been formulated [31].

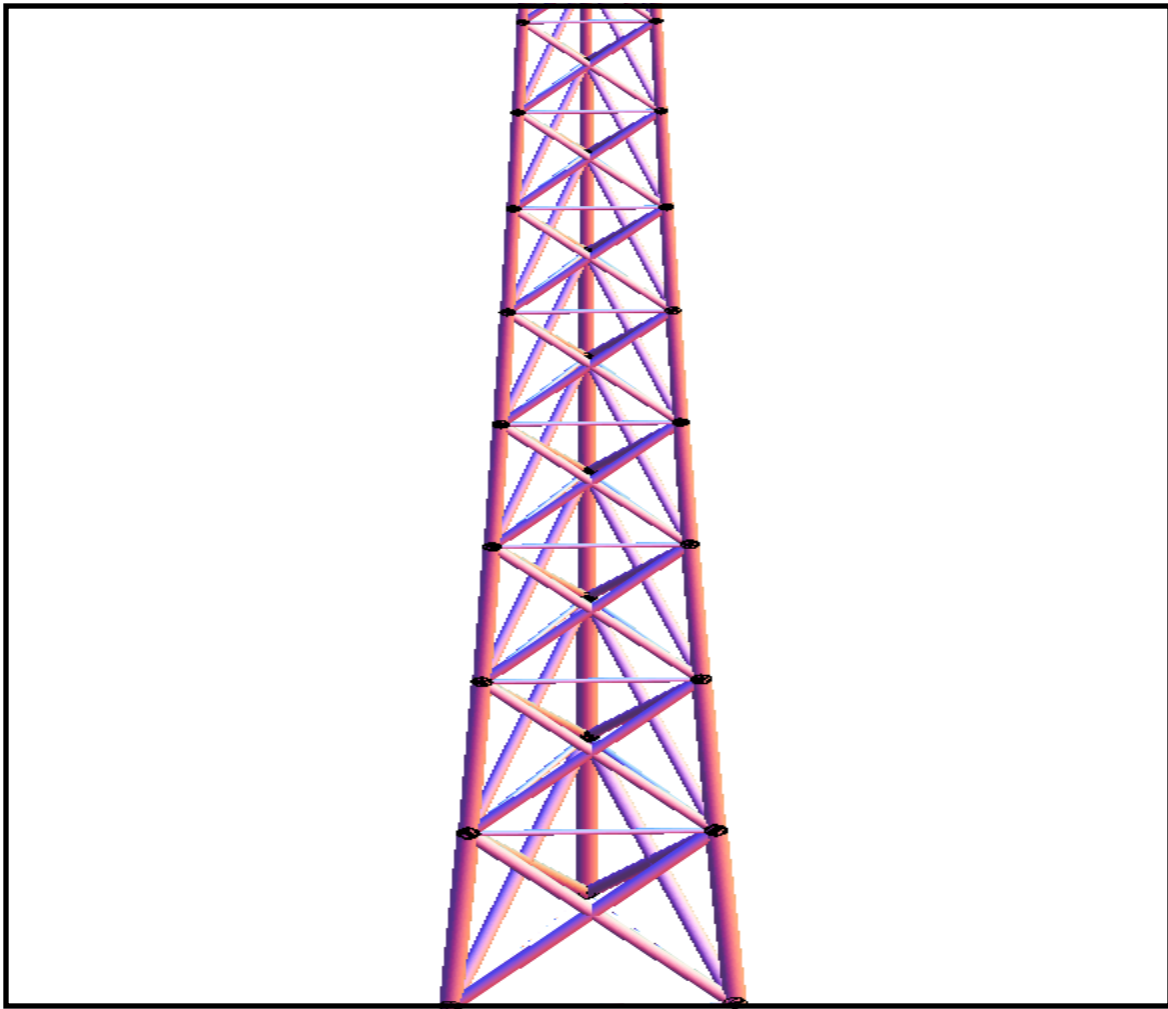


Figure 2.7 Example Portion of Tower with Tubes (rigid structure) drawing with CAD System-AutoCAD.

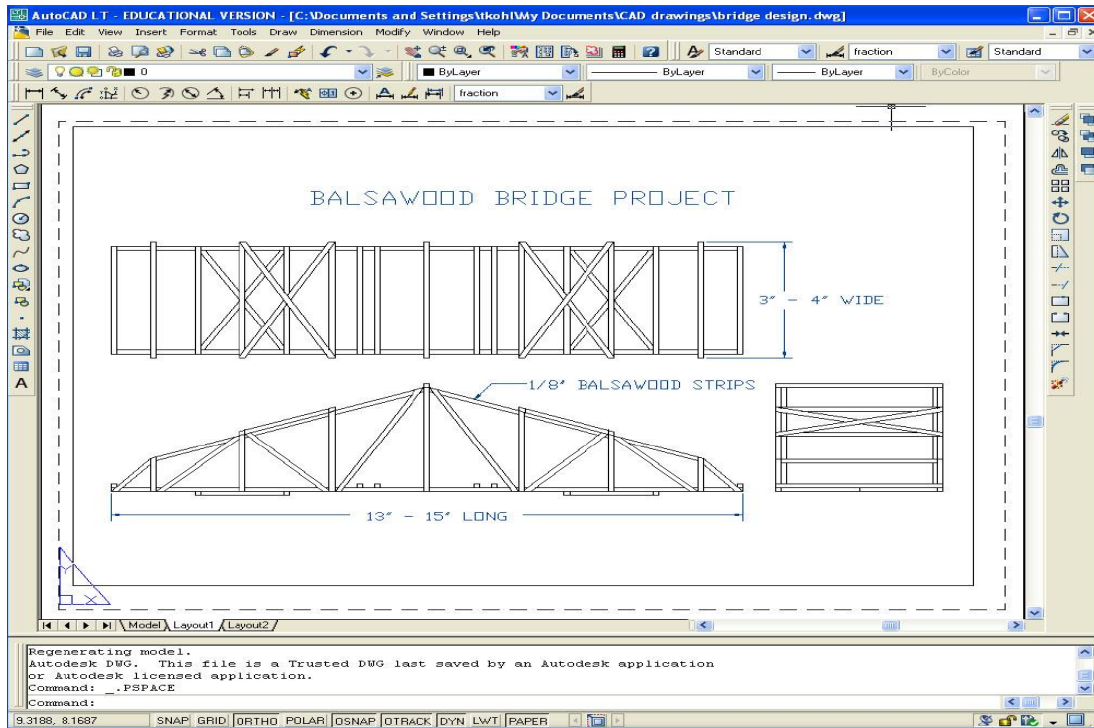


Figure 2.8 Rigidity Theory of Bridge in domain of Architecture

In this chapter, we have seen some applications of rigidity theory, molecular biology, sensor network, CAD. Without forgetting the mechanical linkage.

CHAPTER 3

THE PEBBLE GAME ALGORITHM

1. Introduction

In [10], Jacobs and Hendrickson proposed an efficient algorithm, called pebble game, to check if a given graph is rigid. The pebble game algorithm takes only $O(nm)$ time, where n is the order and m is the size of the given graph. In this chapter we give and describe the pebble game algorithm proposed in [10].

2. The pebble game algorithm

The following two theorems are foundations of the pebble game algorithm.

Theorem 3.1 [15] The edges of a graph $G = (V, E)$ are independent in $\mathcal{R}^2$ if and only if no subgraph $G' = (V', E')$ of G has more than $2n' - 3$ edges, where n' is the order of G' .

Theorem 3.2 [15] For a graph $G = (V, E)$ of order n and size m , the following statements are equivalent.

A. The edges of G are independent in $\mathcal{R}^2$.

B. For each edge (a, b) in G , the graph formed by adding three additional (multiple) edges (a, b) has no induced subgraph G' in which $m' > 2n'$, where n' is the order and m' is the size of G' .

Let G be a graph of order n and size m . The basic idea behind the pebble game algorithm is to grow a maximal set of independent edges at a time. Denote these basis edges (the edges that belong to a maximal set of independent edges) by $\hat{E}$. A new edge can be added to $\hat{E}$ if it is discovered to be independent of $\hat{E}$. If $2n-3$ independent edges are found, then G is rigid. The key is the efficient determination of whether or not a new edge is independent to the current basis $\hat{E}$.

Assume that we have a set $\hat{E}$ of independent edges. Combine $\hat{E}$ with the vertices of G and thus form a graph $\hat{G}$. We want to determine if another edge e is independent of $\hat{E}$ by adding e to $\hat{G}$. Let $\bar{G}$ be the graph obtained by adding e to $\hat{G}$. By Theorem 3.2, e is independent of $\hat{E}$ if and only if there is no subgraph with too many edges

such that $m' > 2n'$ after any edge in $\bar{G}$ is quadrupled (i.e., adding three edges between the same pair of vertices). The following lemma suggests that only e needs to be quadrupled.

Lemma 3.3 [10] A new edge e is independent of $\hat{E}$ if and only if the graph G_{4e} formed by quadrupling e has no induced subgraph G' in which $m' > 2n'$, where n' is the order and m' is the size of G' .

Lemma 3.3 reduces the time complexity of independence testing to that of counting edges in subgraphs once a new edge is quadrupled. The pebble game algorithm is based

on Lemma 3.3 and it works as follows.

2.1.The begin of the pebble game algorithm

The first step of any pebble game algorithm is to assign 'pebbles' to each vertex of the graph (Figure 3.2(1)). The pebbles are representations of degrees of freedom, where each pebble represents a single degree of freedom. In our case we will be assigning two pebbles to each vertex of the multigraph G (remember that each vertex is a fully rigid body which has two degrees of freedom). So initially, the multigraph G has a total of $2|V|$ pebbles lying on the vertices of G (we imagine that no edges are present at this moment). The pebble game algorithm proceeds by moving (or placing) the pebbles from vertices onto incident edges (edges constrain the motion – remove degrees of freedom) (Figure 3.2(2)). The idea is to cover as many edges by pebbles as possible, where each edge gets at most one pebble. When an edge becomes covered by the pebble, the edge is declared independent, which is the independence condition in Lemma 3.3. When we cover an edge (place a pebble on to the edge), the corresponding edge becomes a directed edge. We direct the edge away from the incident vertex where the pebble came from. Specifically, if edge $e = (a, b)$ is covered by a pebble from vertex a , then the edge e becomes directed from a to b . In this sense, the pebble game is constructing a directed multigraph. As we can already imagine, the pebble game algorithm is very attractive for its visual appeal. All of these ideas and concepts will make a lot more sense once we actually give the pseudocode, and when we visually explore it with examples.

As we have originally started with $2|V|$ pebbles (2 pebbles per vertex), it is important to note that throughout the pebble game algorithm the entire system (edges and vertices) will always have $2|V|$ pebbles. Some of these pebbles will be free pebbles (remain on the vertices) while others will be used to cover the incident edges, giving those edges a preferred direction. It is useful to say that a pebble is associated with vertex $v \in V$ if the pebble is either a free pebble on v or it is covering an incident edge at v , and directing that edge out of v . Since pebbles can only cover incident edges. We will also show that throughout the pebble game algorithm that all two pebbles which are associated with v , can always be returned to v as free pebbles, as they correspond to its two trivial degrees of freedom (motions of a rigid body).

As we continue to cover edges (remove free pebbles – add independent edges), making sure that before we cover an edge its ends have at least three free pebbles, it will quickly become apparent that we will find ourselves in the situation where we do not have three free pebbles on

the ends of the edge that we are trying to cover (i.e. test for independence). This happens when the pebbles from the endvertices of the edge at interest are currently being used to cover other incident edges. To deal with this scenario, we need to introduce two important pebble game operations: swap and cascade.

Assume we are given an edge $e = \{v, w\}$ which is covered by a pebble (declared independent), and that e is directed out of v . We also suppose that w has at least one free pebble. A swap is a move in the pebble game where the pebble on w is placed on the edge e , and the pebble that was on e is returned to v . So, under the swap, three things have changed: the number of free pebbles on v is increased by one, the number of free pebbles on w is decreased by one, and e is now directed out of w , where the former change is the most important.

The following is an approach for moving pebbles to cover a new edge. Assume that we have a set of edges $\hat{E}$ that are covered with pebbles. First, look at the two vertices incident to the new edge. If either one of the two vertices has a free pebble, then use it to cover the new edge and exit. Otherwise, their pebbles are used to cover existing edges. If a vertex at the other end of one of these edges has a free pebble, then that pebble can be used to cover the existing edge, freeing up a pebble which can be used to cover the new edge. More precisely, the algorithm searches for free pebbles in a directed graph, in which each edge has a direction. We list in Figure 3.2 the pebble game algorithm given in [10].

2.2.The end of the pebble game algorithm

If Enlarge Cover fails, then all the pebbles owned by the vertices encountered in the search are already covering bonds. If n' vertices were encountered, then they must have at least $2n'$ induced edges to spend all their pebbles. This observation proves that the existence of a pebble covering is equivalent to the edge independence, see the following.

```

Algorithm Enlarge_Cover( $G, e_{ab}$ )
  For Each vertex  $v$ 
     $\text{seen}(v) = \text{False}$ ,  $\text{path}(v) = -1$ 
   $\text{found} = \text{Find\_Pebble}(G, a, \text{seen}, \text{path})$ 
  If ( $\text{found}$ ) Then
     $\text{Rearrange\_Pebbles}(G, a, \text{seen})$ 
    Return (Success)
  If (not  $\text{seen}(b)$ ) Then
     $\text{found} = \text{Find\_Pebble}(G, b, \text{seen}, \text{path})$ 
    If ( $\text{found}$ ) Then
       $\text{Rearrange\_Pebbles}(G, b, \text{path})$ 
      Return (Success)
  Return (Failure)

Function Find_Pebble( $G, v, \text{seen}, \text{path}$ )
   $\text{seen}(v) = \text{True}$ ,  $\text{path}(v) = -1$ 
  If ( $v$  has free pebble) Then
    Return (True)
  Else
     $x = \text{neighbor along edge my pebble covers}$ 
    If (not  $\text{seen}(x)$ ) Then
       $\text{path}(v) = x$ 
       $\text{found} = \text{Find\_Pebble}(G, x, \text{seen}, \text{path})$ 
      if ( $\text{found}$ ) Then Return (True)
     $y = \text{neighbor along edge my other pebble covers}$ 
    If (not  $\text{seen}(y)$ ) Then
       $\text{path}(v) = y$ 
       $\text{found} = \text{Find\_Pebble}(G, y, \text{seen}, \text{path})$ 
      if ( $\text{found}$ ) Then Return (True)
  Return (False)

Subroutine Rearrange_Pebbles( $G, v, \text{path}$ )
  While ( $\text{path}(v) \neq -1$ )
     $w = \text{path}(v)$ 
    if ( $\text{path}(w) = -1$ ) Then
      Cover edge  $(v, w)$  with free pebble from  $w$ 
    Else
      Cover edge  $(v, w)$  with pebble from edge  $(w, \text{path}(w))$ 
     $v = w$ 

```

Figure 3.1 Algorithm for enlarging a pebble covering.

Lemma 3.4 [10] If the new edge e is tripled instead of quadrupled to form G_{3e} , then G_{3e} has a pebble covering.

Lemma 3.5 [10] If G_{4e} , the graph constructed by quadrupling e , does not have a pebble covering, then the set of n' vertices encountered in Find Pebble already have $2n' - 3$ induced edges.

Therefore, if the pebble game algorithm determines that a new edge is not independent, then it means that a subgraph on n' vertices already contains $2n' - 3$ edges; otherwise, the pebble game algorithm will enlarge the covering successfully.

Theorem 3.6 [10] A new edge e is independent of $\hat{E}$ if and only if there exists a pebble covering when e is quadrupled.

Corollary 3.7 [10] If $\hat{E}$ is independent and a corresponding pebble covering $\hat{G}$ is known, then determining whether a new edge is independent requires $O(n)$ time.

The pebble game algorithm takes $O(nm)$ time, in which each enlargement of a pebble covering requires $O(n)$ time.

2.3.The pebble game algorithm proceeds

In Figure 3.2(this example is a demonstration presented by KINARI[29]), $2|V|-3$ pebble game algorithm[31]. We want to test the graph for rigidity/flexibility. We start by placing two pebbles on each vertex (Figure 3.2 (1)). We test edges one by one. The edge that is currently tested is highlighted in red. If we have four free pebbles on the ends of the edge, we can pebble that edge (i.e. that edge is declared independent), and we direct that edge.

Continued. In (Figure 3.2 (6)) the edge that is being tested has only three free pebbles on its ends. We perform a swap with the neighboring edge (Figure 3.2 (6)) and the fourth free pebble appears, and we can now pebble that edge (Figure 3.2 (8)). We continue to test and pebble more edges (Figure 3.2 (9)). So We continue to test and cover edges one by one (Figure 3.2(10), (11)). Indicates the graph is minimally rigid as it has three remaining free pebbles.

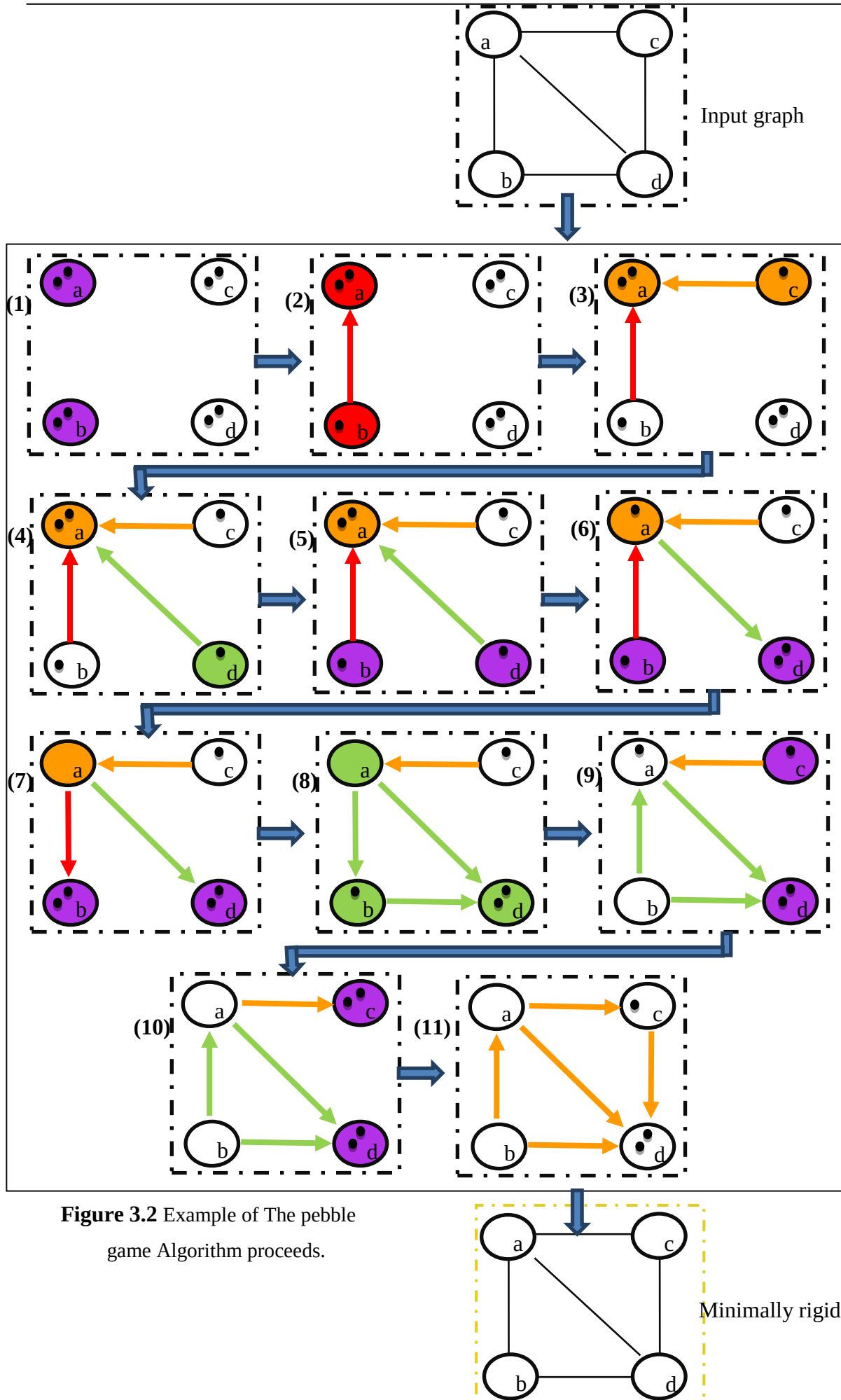


Figure 3.2 Example of The pebble game Algorithm proceeds.

CHAPTER 4

IMPLEMENTATION

1. introduction

After establishing a conceptual study of testing rigidity using the pebble game algorithm, in this chapter, we will discuss the implementation of our algorithm that take up in the previous chapter, we divided this implementation in two conceptually parts, first part is Algorithm implementation, and the second part is a visual (graphic) representation.

2. Implementation

2.1.Part 1: Algorithm implementation

We realized our algorithm implementation with the Java programming language using the IDE (integrated development environment) NetBeans 8.0.2, because is an effective way to present the algorithms and data structures as real programs.

We represent a graph as a set of vertices and a set of edges, and we represent them as an Boolean matrix (it has a 1 (true) in row v and column w if there is an edge connecting vertex v and vertex w and a 0 (false) in row v and column w if there is no such edge.). in Figure 4.1 (2) we represent example of a triangle ABC, and in Figure 4.1 (1) is an implementation of the triangle graph that uses a undirected representation of matrix, built as an array of arrays.

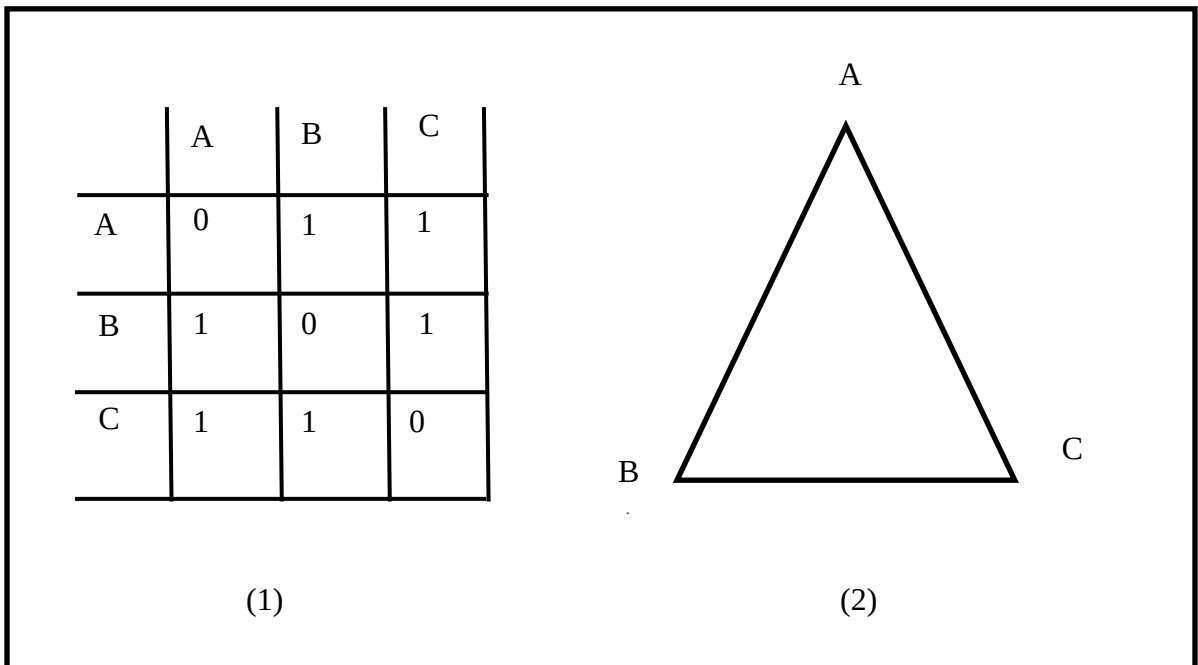


Figure 4.1 (1) undirected representation of matrix, (2) example of a triangle

Also we represented the undirected graph with java code uses adjacency matrix to this way:

```
public class Graph {
    private Vertex vertexList []; // array of vertices
    int adjMat [] []; //Edges will be represented as adjacency Matrix
    int nVerts; // current number of vertices
    public Graph () {
        vertexList = new Vertex[MAX_VERTS];
        adjMat = new int[MAX_VERTS] [MAX_VERTS];
        nVerts = 0;
        for (int j=0; j<MAX_VERTS; j++) // set adjacency matrix to 0
            for (int k=0; k<MAX_VERTS; k++)
                adjMat[j][k] = 0;
    }
    public void addVertex (String lab) // argument is label
    {
        vertexList[nVerts++] = new Vertex(lab);
    }
    //This method will be called to make connect two vertices
    public void addEdge (int start, int end)
    {
        adjMat[start][end] = 1;
        adjMat[end][start] = 1;
    }
}
```

We pass to the pebble game algorithm it containing four functions as follows:

1. Rearrange pebbles: swapped the free pebble is found
2. Find pebble: searching for free pebbles in a directed graph we use a breadth first search
3. Enlarge cover: it's applied four time, once for each copy of the quadrupled edge which can cover the new edge
4. Rigid: counting the total number of pebble in the end of algorithm, If the total number of remaining pebbles is 3, the graph is rigid and we return true. Otherwise, we return false

2.2.Part 2: visual representation

Generally, the part of visualization: it performs the following steps:

2.2.1. Step 1: Read a graph from xml file by reading them with The Document Object Model provides APIs, following are the steps used while parsing a document using DOM Parser:

- Import XML-related packages.

```
import org.w3c.dom.*;
import javax.xml.parsers.*;
import java.io.*;
```

- Create a DocumentBuilder

```
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
```

- Create a Document from a file or stream

```
StringBuilder xmlStringBuilder = new StringBuilder();
xmlStringBuilder.append("<?xml version='1.0'?> <vertex id =\"A\"> </vertex>
<vertex id =\"B\"> </vertex> <edge V1 =\"A\" V2 =\"B\"> ");
ByteArrayInputStream input = new ByteArrayInputStream(
    xmlStringBuilder.toString().getBytes("UTF-8"));
Document doc = builder.parse(input);
```

- Extract the root element

```
Element root = document.getDocumentElement();
```

- Examine attributes

```
//returns specific attribute
getAttribute("Graph");
//returns a Map (table) of names/values
getAttributes();
```

- Examine sub-elements

```
//returns a list of subelements of specified name
getElementsByTagName("Vertex");

//returns a list of all child nodes
getChildNodes();
```

We will be storing a graph in the file using **EX**tensible **M**arkup **L**anguage (XML) as following example we have a representation for triangle ABC:

```
<? xml version="1.0" encoding="UTF-8"?>
<graph>
  <vertex id="A">
    <position x="582" y="330"/>
  </vertex>
  <vertex id="B">
    <position x="480" y="144"/>
  </vertex>
  <vertex id="C">
    <position x="696" y="144"/>
  </vertex>
  <edge v1="2" v2="1"/>
  <edge v1="0" v2="2"/>
  <edge v1="1" v2="0"/>
</graph>
```

2.2.2. Step 2: after reading the xml file after choosing with we pass to draw the graph in the panel we using the library jGraph[30], it is allowing display vertices and their relations(edges) by drawing individual elements. Each element displayed by the graph contains exactly one item of data, which is called a cell. A cell may either be a vertex, an edge or a port. Vertices have zero or more neighbours, and edges have one or no source and target vertex. Each cell has zero or more children, and one or no parent.

The following code creates a JGraph object:

```
JGraph graph = new JGraph ();

...

JScrollPane scrollPane = new JScrollPane(graph)
```

The code creates an instance of JGraph and puts it in a scroll pane. JGraph's constructor is called with no arguments in this example. Therefore, JGraph will use an instance of DefaultGraphModel, create an instance of GraphView for it, and add a sample graph to the model [30].

JGraph offers the following forms of interactions:

- In-place editing
- Moving
- Sizing
- Bending (Adding/Removing/Moving edge points)

3. Experimental result

In this section we present a set of test examples of our algorithm

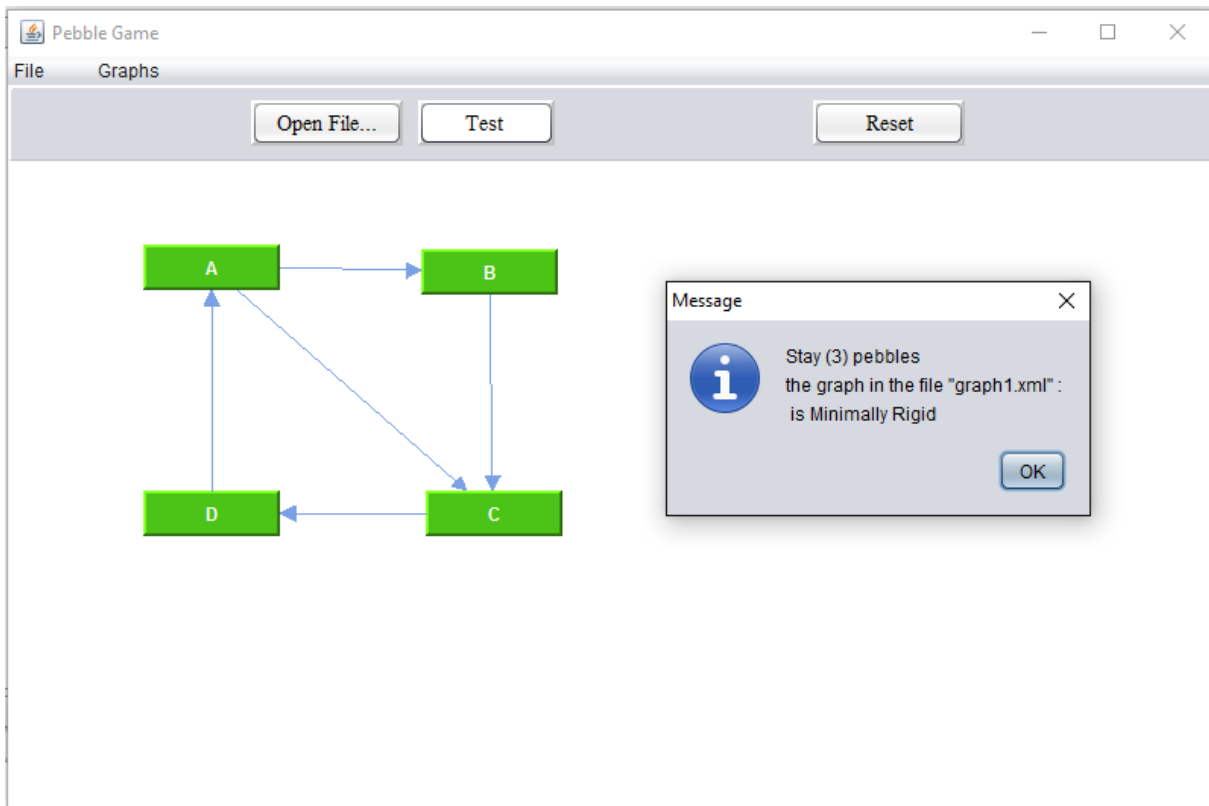


Figure 4.2 result of test for graph with four vertices

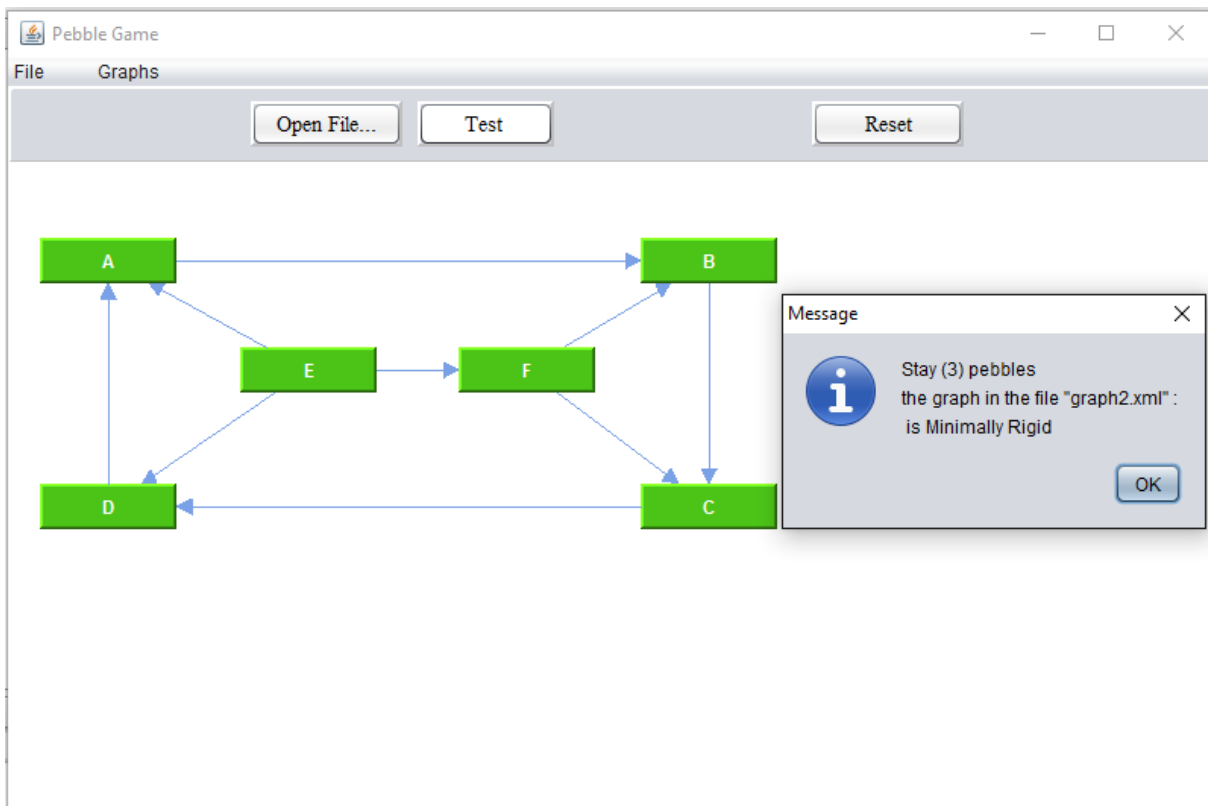


Figure 4.3 result of test for graph with six vertices

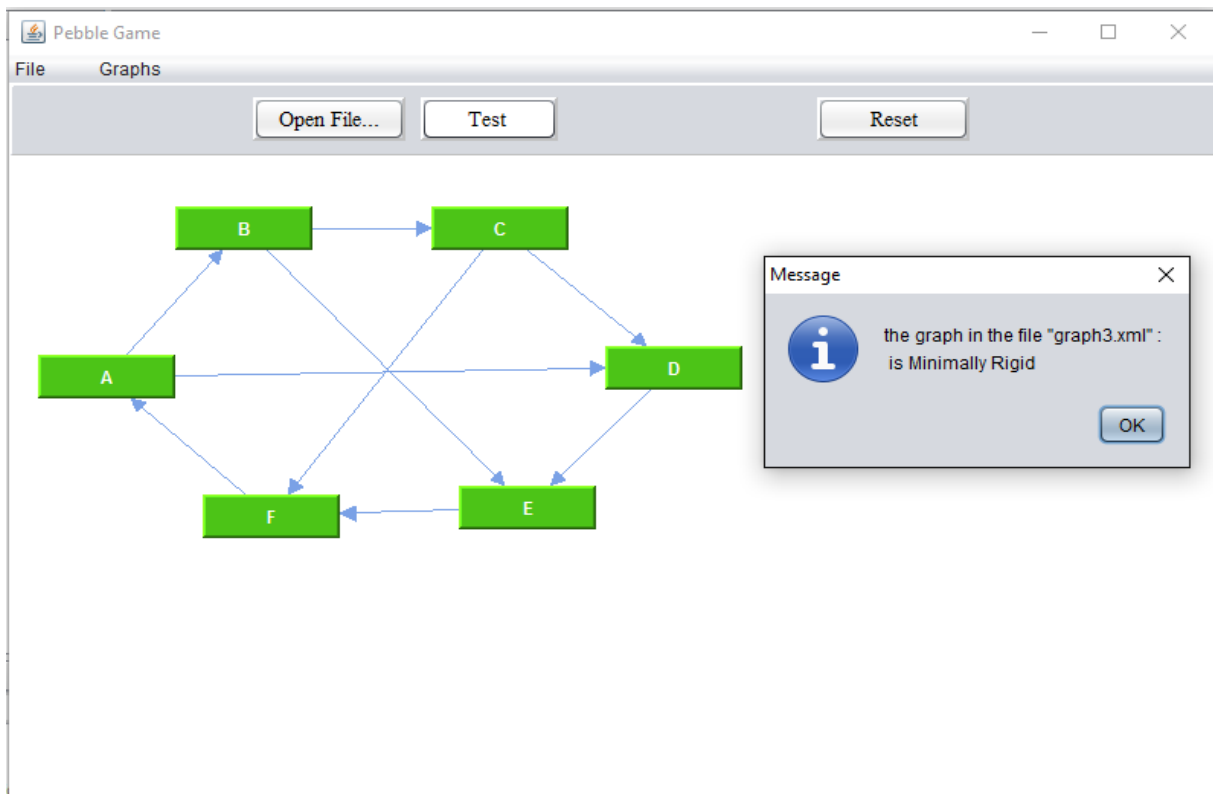


Figure 4.4 result of test for graph with six vertices

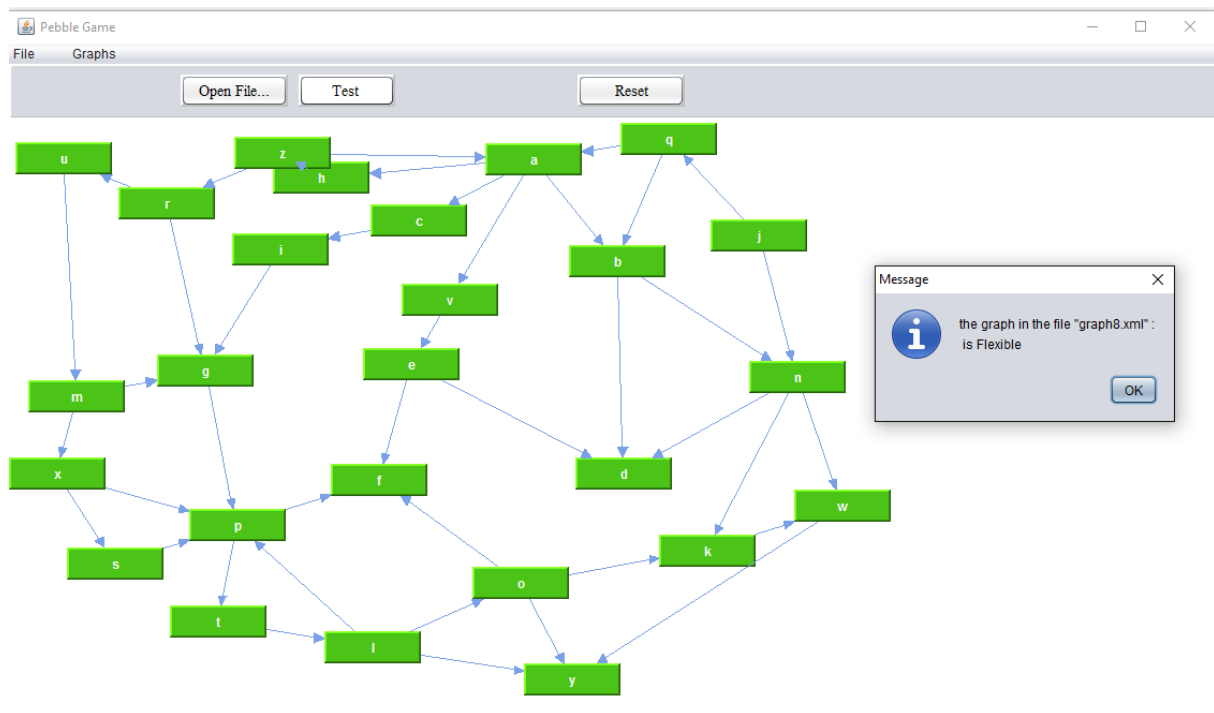


Figure 4.5 result of test for graph with twenty-six vertices

In this chapter, we seen the centralized implementation of the pebble game algorithm using java t and the needed tools and components for achieving the goal. we've verified our algorithm and it worked.

GENERAL CONCLUSION

General Conclusion

The rigidity theory gives a beautiful and complete combinatorial characterization of rigidity and flexibility of bar and joint frameworks in 2 dimension. different applications related to rigidity theory have been made in several fields. This theory has facilitated the development of the pebble game algorithms and it is an efficient algorithm for studying flexibility and rigidity and it can be used to explore new and interesting questions which will lead to further research possibilities. We also implement a computer program for the algorithm that Jacobs and Hendrickson has proposed in [10].

In conclusion, for future work, to increase the performance of our computer program we want to

- check if a given graph is rigid, redundantly rigid or globally rigid.
- extract rigid and flexible regions from a given graph.

References

- [1] B. Anderson, D. O. P. N. Belhumer, T. Eren, A. S. Morse and W. Whiteley, Operations on rigid formations of autonomous agents, *Communications in Information and Systems*, Vol. 3, No. 4, p. 223–258, 2004.
- [2] Andrusier N, Mashiach E, Nussinov R, et al. Principles of flexible protein-protein docking. *Proteins* 2008; 73:271–89.
- [3] J. Aspnes and T. Eren and D. Goldenberg, et al., “A theory of network localization,” *IEEE Transactions on Mobile Computing*, vol. 5, no. 12, 2006.
- [4] T. Eren, W. Whiteley, B. D.O. Anderson, A. S. Morse, P. Belhumeur, Information Structures to Secure Control of Rigid Formations with Leader-Follower Structure, In *Proc. of the American Control Conference*, pp. 2966-2971, Portland, Oregon, 2005.
- [5] Fontana A, Filippis V, Laureto PP. Rigidity of thermophilic enzymes. *Progress in Biotechnology* 1998; 15:277–94
- [6] Fox N, Jagodzinski F, Li Y, et al. KINARI-Web: a server for protein rigidity analysis. *Nucleic Acids Res* 2011; 39:W177–83.
- [7] J. Graver, *Counting on Frameworks: Mathematics to Aid the Design of Rigid Structures*, The Mathematical Association of America, Washington, 2001.
- [8] J. Graver, B. Servatius, and H. Servatius, *Combinatorial Rigidity*, Graduate Studies in Math., AMS, 1993.
- [9] H. Gluck, almost all simply connected closed surfaces are rigid, In *Geometric Topology*, Lecture notes in Mathematics, Springer-Verlag, Berlin, No. 438, p. 225–239, 1975.
- [10] T. He, C. Huang, B. Blum, J. Stankovic, and T. Abdelzaher, “Range-free localization schemes in large scale sensor networks,” in *Proc. Ninth Int’l Conf. Mobile Computing and Networking (MobiCom)*, pp. 81-95, 2003.
- [11] B. Hendrickson and D.J. Jacobs, An algorithm for two dimensional rigidity percolation: The pebble game, *J. Comput. Phys.*, 137:346-365, 1997.
- [12] B. Hendrickson, Conditions for unique graph realizations, *SIAM J. Comput.*, 21, pp. 65-84, 1992.
- [13] Adnan Sljoka, *Counting For Rigidity, Flexibility And Extensions Via The Pebble Game Algorithm*, Master Degree Thesis, York University, Toronto, Ontario, September 2006

- [14] Jacobs DJ, Rader AJ, Kuhn LA, et al. Protein flexibility predictions using graph theory. *Proteins: Struct Funct Genet* 2001; 44:150–65.
- [15] L.A. Kuhn, D.J. Rader and M.F. Thorpe, Protein flexibility predictions using graph theory, *Proteins*, 44:150-65, 2001.
- [16] G. Laman, “On graphs and rigidity of plane skeletal structures,” *J. Eng. Math.*, vol. 4, pp. 331-340, 2002.
- [17] A. Mantler and J. Snoeyink, Banana Spiders: A study of connectivity in 3D combinatorial rigidity, *CCCG*, 44-47, 2004.
- [18] D. Moore, J. Leonard, D. Rus, and S.Teller, “Robust distributed network localization with noisy range measurements,” in *Proc. Second ACM Conf. Embedded Networked Sensor Systems (SenSys)*, Nov. 2004.
- [19] A. Recski, *Matroid Theory and its Applications*, Springer-Verlag, Berlin, 1989.
- [20] A.Savvides, C.-C. Han, and M.B. Strivastave, “Dynamic fine-grained localization in ad-hoc networks of sensors,” in *Proc. Seventh Int’l Conf. Mobile Computing and Networking (MobiCom)*, pp. 166-179, 2001.
- [21] B. Servatius and W. Whiteley, (1995). Constraining plane configurations in CAD: Combinatorics of directions and lengths, Preprint.
- [22] A. Sljoka, O. Shai. and W. Whiteley, Checking Mobility and Decomposition of Linkages via Pebble Game Algorithm, *Proceedings for ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, IDETC/CIE*, Aug 28-31, 2011, Washington, USA, 2011.
- [23] K. Sugihara, (1985). Detection of structural inconsistency in systems of equations with degrees of freedom, *J. Disc Appl. Math.* 10, 297–312.
- [24] Thomas S, Tang X, Tapia L, et al. Simulating protein motions with rigidity analysis. *J Comput Biol* 2007; 14:839–55.
- [25] M.F. Thorpe, D.J. Jacobs, N.V. Chubynsky and A.J. Rader, Generic rigidity of network glasses. In: Thorpe MF, Duxbury PM, eds. *Rigidity theory and applications*. New York: Kluwer Academic/ Plenum Publishers, pp 239-277, 1999.
- [26] M. F. Thorpe, *Rigidity percolation Physics of Disordered Materials (Institute for Amorphous Studies Series)* (New York: Plenum), 1985.
- [27] W. Whiteley, Counting out to the flexibility of molecules, *Phys. Biol.* 2, S116-S126, 2005.

- [28] W. Whiteley, Some Matroids from discrete applied geometry. In J. Bonin, J.Oxley, and B. Servatius, editors, Matroid Theory, volume 197 of Contemp. Math., pages 171-311. Amer. Math. Soc., Providence, 1996.
- [29] Merse E. Gáspár and Peter Csermely . Rigidity and flexibility of biological networks. Briefings in Functional Genomics (2012) 11: 443-456.
- [30] Sung-Yu Tsai , The Rigidity Property and the Unique Localization Problem of Sensor Networks, Master Degree Thesis, Hsinchu, Taiwan, Republic of China, January 2010.
- [31] Brigitte Servatius. The Geometry Of Frameworks: Rigidity, Mechanisms And Cad

Websites

- [29] <http://kinari.cs.umass.edu>, consulted the: 05/02/2016.
- [30] <https://www.jgraph.com>, consulted the: 04/21/2016.
- [31] <http://linkage.cs.umass.edu/pg/>, consulted the: 02/15/2016

Abstract

In this dissertation, we start our study with rigidity theory that can test rigidity and flexibility of bar and joint frameworks in 2 dimension. different applications related to rigidity theory have been made in several fields. This theory has facilitated the development of the pebble game algorithms, and we implement a computer program to test the rigidity and flexibility of a given graph.

Key Words: rigidity theory, rigidity, bar and joint frameworks, 2 dimension, pebble game algorithms, graph.

Résumé

Dans cette thèse, nous commençons notre étude avec la théorie de la rigidité. Cette théorie peut tester la rigidité et souplesse du bar and joint frameworks dans 2 dimensions. Des applications différentes reposent sur la théorie de la rigidité ont été faites dans plusieurs domaines. Cette théorie a facilité le développement des algorithmes de pebble game, et nous avons implémenté un programme d'ordinateur pour tester la rigidité et la flexibilité d'un graphe donné.

Mot clés : la théorie de la rigidité, la rigidité, 2 dimension, bar and joint frameworks, les algorithmes de jeu de galets, graphe.

ملخص:

في هاته المذكرة، بدأنا دراستنا بنظرة عامة حول نظرية الصلابة، هذه النظرية التي تمكّن من اختبار قوة ومرونة الأشكال في الأبعاد الثنائية، تعتمد عدة تطبيقات على نظرية الصلابة والتي طُبقت في عدة مجالات، وقد سهلت بتطوير خوارزمية تسمى بلعبة الحصى، حيث قمنا بتطبيق برنامج كمبيوتر يسمح باختبار صلابة ومرونة شكل هندسي معين.

الكلمات المفتاحية: نظرية الصلابة، الصلابة، البعد الثنائي، خوارزمية لعبة الحصى.