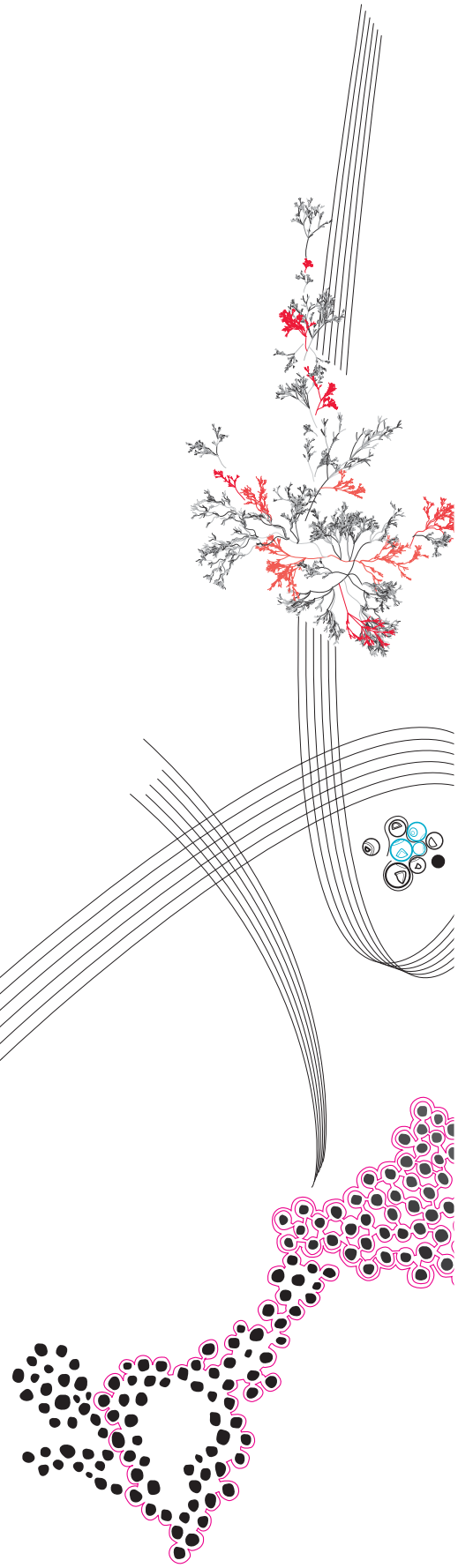

BSc Thesis Applied Mathematics

An algebraic geometry approach to three-dimensional graph rigidity

Kennet Libohova (s1995006)

Supervisor: Frederic P. Schuller

June 26, 2020

Department of Applied Mathematics
Faculty of Electrical Engineering,
Mathematics and Computer Science

UNIVERSITY OF TWENTE.

Keywords: Graphs, rigidity

This report analyses a special topic in the field of graph theory: graph rigidity. Specifically, three dimensional graph rigidity, for which necessary and sufficient conditions, which at the same time are practically workable, still are to be found. This issue corresponds to the graph rigidity problem being conjectured as an NP-hard problem. We will define graph rigidity and flexibility, both locally and globally. We will then create two specific maps acting on the nodes of the graph. This map will yield a system of an equation and an inequality which will describe the rigidity conditions of a given graph. Last, using tools available in Wolfram Mathematica, conclusions on the local/global rigidity of a graph will be drawn based on the output of the algorithm.

Contents

I. INTRODUCTION

This report explores an approach to answering the question whether a graph representation [1] is rigid or not in the three-dimensional case. A graph representation is a pair (G, p), where $G = (\mathcal{V}, E, \phi)$ is a graph defined by a set of nodes $\mathcal{V}$, a set of edges E and an incidence function ϕ which maps each edge to the unique couple of nodes it connects (if any, these nodes are called adjacent) and p is a map from $\mathcal{V}$ to $\mathbb{R}^d$ (for a positive integer dimension d) such that for all $u, v \in \mathcal{V}$ one has $p(u) = p(v)$ only if $u = v$. The set $p(\mathcal{V})$ is then the embedding of the nodes in $\mathbb{R}^d$. We will be concerned only with graph representations in this thesis.

Intuitively speaking, we say a graph representation is *rigid* if there is no move that can be done to any vertices so that the distance of connected vertices remains the same while the distance between non-connected vertices is changed. Conversly, a graph representation is said to be flexible. For a better understanding of this concept consider figures 1 and 2. In figure 1, we can move the two top vertices of rectangle to the right to obtain the

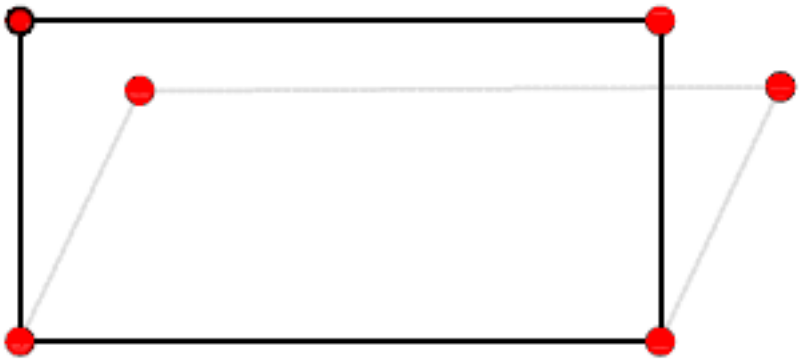

FIGURE 1: Non-rigid in $d = 2$.

parallelogram which shares two vertices with the rectangles. The parallelogram has the same distance of adjacent points as the initial one, but a different length of the diagonals, which however are not contained in the set E of distances to be preserved. Thus, the graph representation is flexible. On the other hand, figure 2 presents a rigid graph: the move we discussed before is undoable in this case in a two dimensional plane, given that now the diagonal is part of the set E of edges whose distance must not change. There is actually a move that could be done to deform its shape while preserving the lengths of the edges

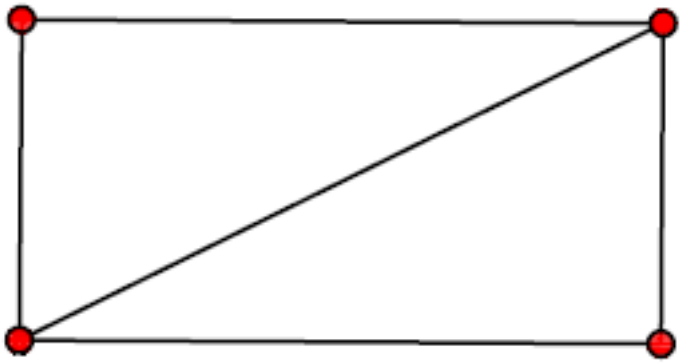

FIGURE 2: Rigid in $d = 2$.

in E, but that still keeps the graph rigid, informally speaking. We will see in the next paragraph what we mean by this.

It is useful to distinguish two different kinds of rigidity a graph representation might have: local and global. Considering figures 3, 4 and 5 we see flexible, a locally rigid and a globally rigid graph, respectively. We consider all three graphs in $d = 2$ for explaining the idea. We see that figure 4 -as figure (2)- is "sort of rigid", with the exception of two movements of two points which change its distance, namely, mirroring either the top right or bottom left point with respect to the diagonal would preserve adjacent nodes distances but change the distance between the two non-adjacent points. Speaking a little bit more

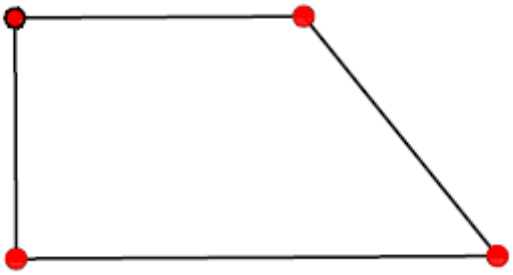

FIGURE 3: flexible graph in $d = 2$.

precisely, a graph representation is said to be locally rigid if we can move some of its vertices and change non adjacent vertices distances, but we cannot do it with a continuous move. We can conclude that global rigidity is a stronger feature than local rigidity.

The rigidity problem has been solved for $d < 3$: by means of advanced combinatorics

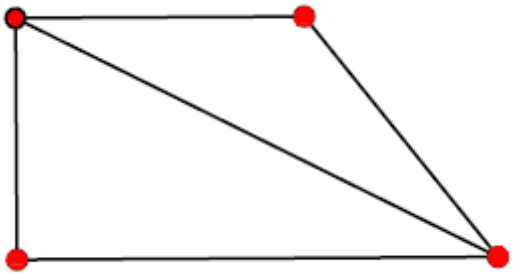

FIGURE 4: locally rigid graph in $d = 2$.

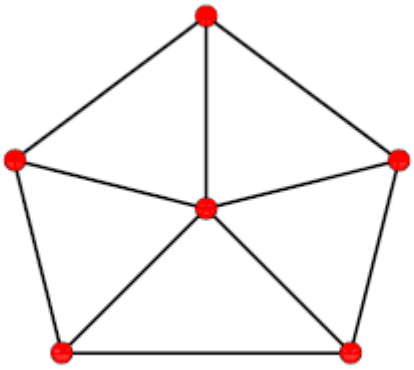

FIGURE 5: globally rigid graph in $d = 2$.

techniques this decision problem in dimensions 1 and 2 can be done efficiently. For $d \geq 3$ this is not the case. The decision problem in higher dimensions is still open and there is a feeling among scientists that this is a NP-hard problem [1]. Finding an efficient algorithm for this decision problem would have outstanding consequences: for starters, if it could be proved to be NP-hard, then the millenium problem P=NP would be solved. Moreover, for more concrete applications, being able to determine whether a representation is rigid or not would be of huge use and interest to engineering related fields, robotics, chemistry or biology, to cite some.

In this report we will analyse the specific case $d = 3$ and develop a theory which theoretically allows us to answer the decision problem. More specifically, in Section 2 we will present formally the problem and technically derive a system of quadratic multivariate equations and inequalities that describe the rigidity condition of the representation. In Section 3 we will explain why the problem as formulated can be seen as an application of the

Tarski-Seidenberg theorem and how the Cylindrical Algebraic Decomposition algorithm [2] and the Exists[] function in Wolfram Mathematica come to our advantage for this decision problem. In Section 4 we will see some concrete examples, solved either by intuitive reasoning and trial and/or with the use of the algorithms. Finally, in Section 5 we will give a conclusion.

II. THEORETICAL CONSTRUCTIONS

In this section we will formulate the problem more formally for the three-dimensional case. Then we will introduce two maps and, by means of linear algebra and standard inner product techniques, we will reformulate once more the problem as a system of a multivariate quadratic equation and inequality with one unknown vector. After that, we will prove some properties of the maps introduced.

A. Precise formulation of the problem

In this subsection we will precisely define global and local flexibility of a graph representation. The reason for choosing to define flexibility rather rigidity is related to the whole analysis that has been done on this problem.

As we have said previously, a graph representation is a couple (G, p) formed by a graph $(\mathcal{V}, E, \phi)$ and a map p. In dimension $d = 3$ we can interpret this map as being the Eucledian coordinates of each node. For convenience, from now on, we will refer to the graph representation as just a graph. Moreover, we will refer to p as already the map outcome, i.e., the coordinate vector of a certain node.

To be more specific, let $G(\mathcal{V}, E, \phi)$ be a graph embedded in $\mathbb{R}^3$, that is, a graph with position vectors $p_k \in \mathbb{R}^3$ $(k = 1, 2, ..., \nu)$ where we assign each node a number and ν is the cardinality of the set of vertices $\mathcal{V}$. Also, let $\mathcal{E}$ be the set $\mathcal{E} = \{(i, j)|1 \leq i \leq j \leq \nu\}$ of all possible edges in a given graph and E be the set of present edges in that graph. Clearly $E \subseteq \mathcal{E}$. We have now all the elements for providing the following definitions:

Def. 1: A graph representation is **globally flexible** if there exists a displacement vector $\delta = (\delta_1, \delta_2, ..., \delta_\nu)^T$ such that:
(i) $||(p_i + \delta_i) - (p_j + \delta_j)|| = ||p_i - p_j||$ for all couples $(i, j) \in E$
(ii)$||(p_i + \delta_i) - (p_j + \delta_j)|| \neq ||p_i - p_j||$ for at least one $(i, j) \in \mathcal{E} \backslash E$

Def. 2: A graph representation is **locally flexible** if for any $\epsilon > 0$ there exists a displacement vector δ such that conditions (i) and (ii) in Def.1 hold and in addition it holds $\sum_{i=1}^{\nu} ||\delta_i|| < \epsilon$, i.e., the vector can be chosen arbitrarily small.

When these two definitions do not hold, namely, there is not such displacement vector that can be found, we say the graph representation is globally rigid. If instead there exists one (or more) displacement vector which, however, cannot be chosen arbitrarily small, then the graph representation is locally rigid. As a matter of fact, by a close look to the definitions we see that globally flexible is equivalent to locally rigid: there is one displacement vector for which the definition holds but it can not be chosen arbitrarily small. In other words this means that we can't move one or more vertices continuously so that the distance of connected nodes is preserved and the distance of non-connected nodes is not. It also follows from the definitions that local flexibility $\implies$ global flexibility.

Finding such δ is a quite complicated task, this is why in the next subsection we will try to find write the two conditions in a form which is equivalent but easier to solve for a computer.

B. The relative distance and adjacency maps

In this subsection we will introduce two maps which will allow us to represent the conditions of definition 1 as a system of one multivariate quadratic equation and one multivariate quadratic inequality in the unknown vector δ we are looking for.

We start by representing the set of nodes with respective position vectors as a single column vector $p = (p_1, p_2, ..., p_\nu)^T \in \mathbb{R}^{3\nu}$, where each $p_i \in \mathbb{R}^3$. Let us now introduce the relative position map $r : \mathbb{R}^{3\nu} \to \mathbb{R}^{\frac{3}{2}\nu(\nu-1)}$, which acts on the vector p and maps it to a vector

$$r(p) = ((p_1 - p_2)^{(x)}, (p_1 - p_2)^{(y)}, (p_1 - p_2)^{(z)}, ..., (p_{\nu-1} - p_\nu)^{(x)}, (p_{\nu-1} - p_\nu)^{(y)}, (p_{\nu-1} - p_\nu)^{(z)})$$

which lies in a smaller space and is formed by the relative distances of each node with all the others in all three directions of the Eucledian space. The order in which we place the points will be as follows: we start with node 1 and calculate the relative distances with

nodes 2,3,4,...,ν. Then we continue with node 2 and calculate the relative distance with nodes 3,4,...,ν, and so on until we arrive at node $\nu - 1$.

We can represent the map by the matrix R as follows:

$$R = \begin{bmatrix} I_3 & -I_3 & \mathbf{0} & \dots & \mathbf{0} \\ I_3 & \mathbf{0} & -I_3 & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ I_3 & \mathbf{0} & \mathbf{0} & \dots & -I_3 \\ \mathbf{0} & I_3 & -I_3 & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & I_3 & \mathbf{0} & \dots & -I_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & I_3 & -I_3 \end{bmatrix}$$

where I_3 represents the 3×3 identity matrix and $\mathbf{0}$ stands for the 1×3 zero vector. The map is linear and this will be proved in the next subsection. Notice that we can also apply the map r to the unknown displacement vector δ to obtain the relative displacements.

Taking a closer look now to the equality and inequality that we had in the definition of global flexibility we see we can write them down differently. As a matter of fact, recalling (i) in definition 1:

$$\begin{aligned} &||(p_i + \delta_i) - (p_j + \delta_j)|| = ||p_i - p_j|| \\ &\iff ||(p_i + \delta_i) - (p_j + \delta_j)||^2 = ||p_i - p_j||^2 \\ &\iff (p_i + \delta_i - p_j - \delta_j, p_i + \delta_i - p_j - \delta_j) = (p_i - p_j, p_i - p_j) \\ &\iff (p_i - p_j, p_i - p_j + \delta_i - \delta_j) + (\delta_i - \delta_j, p_i - p_j + \delta_i - \delta_j) = (p_i - p_j, p_i - p_j) \\ &\iff \cancel{(p_i - p_j, p_i - p_j)} + (p_i - p_j, \delta_i - \delta_j) + (\delta_i - \delta_j, p_i - p_j) + (\delta_i - \delta_j, \delta_i - \delta_j) = \cancel{(p_i - p_j, p_i - p_j)} \\ &\iff 2(p_i - p_j, \delta_i - \delta_j) + (\delta_i - \delta_j, \delta_i - \delta_j) = 0 \end{aligned}$$

We recognize the values $p_i - p_j$ and $\delta_i - \delta_j$ as elements of the vectors $r(p)$ and $r(\delta)$ corresponding to the couple of nodes (i, j). This yields:

$$2(r(p)_{i,j}, r(\delta)_{i,j}) + (r(\delta)_{i,j}, r(\delta)_{i,j}) = 0 \tag{1}$$

for the couples (i, j) of adjacent nodes (i.e., edge $e_{i,j} \in E$). Keep in mind that $(\cdot, *)$ represents the standard inner product.

Similarly for condition (ii) of definition 1 we obtain the inequality

$$2(r(p)_{(ij)}, r(\delta)_{ij}) + (r(\delta)_{(ij)}, r(\delta)_{(ij)}) \neq 0 \tag{2}$$

for at least one couple of nodes (i, j) such that $e_{i,j} \in \mathcal{E} \backslash E$.

In order to encode which edges are in E and which in $\mathcal{E} \backslash E$ we will introduce the adjacency matrix A. The idea of the matrix is that it will be zero everywhere but in some entries of the diagonal where it will have three 1s (one per coordinate) each time a couple of points is connected. Remark: the order of the 'node coupling' is the same that we have done for the map r.

For instance consider a rectangle with nodes 1, 2, 3, 4 (set counterclockwise starting from the bottom left node), its adjacency matrix will look as follows:

$$A = \begin{bmatrix} I_3 & \mathbf{0} & \dots & \dots & \dots & \dots \\ \dots & \mathbf{0} & \dots & \dots & \dots & \dots \\ \dots & \dots & I_3 & \dots & \dots & \dots \\ \dots & \dots & \dots & I_3 & \dots & \dots \\ \dots & \dots & \dots & \dots & \mathbf{0} & \dots \\ \dots & \dots & \dots & \dots & \dots & I_3 \end{bmatrix}$$

The entries in the diagonal correspond in order to the pair of points (1,2), (1,3), (1,4), (2,3), (2,4), (3,4).

The matrix A will have by construction dimension $\frac{3}{2}\nu(\nu-1) \times \frac{3}{2}\nu(\nu-1)$. Moreover, it is a Hermitian projector since it can be checked that $A^2 = A$ and that $A^T = A$. This will be done in the next subsection.

We consider (1) and we try to make use of the adjacency matrix and of the matrix R. Since we want (1) to hold for all adjacent nodes, we can write it as:

$$\begin{aligned} & 2(A \cdot R \cdot p, A \cdot R \cdot \delta) + (A \cdot R \cdot \delta, A \cdot R \cdot \delta) = 0 \\ & \iff 2(A \cdot R \cdot p)^T \cdot (A \cdot R \cdot \delta) + (A \cdot R \cdot \delta)^T \cdot (A \cdot R \cdot \delta) = 0 \\ & \iff 2p^T R^T A^T A R \delta + \delta^T R^T A^T A R \delta = 0 \\ & \iff 2p^T R^T A R \delta + \delta^T R^T A R \delta = 0 \end{aligned} \tag{3}$$

where the last implication follows since A is a Hermitian projector, thus $A^T \cdot A = A \cdot A = A^2 = A$.

For equation (2) we have a similar form where instead of A we use $I - A$, with I the $\frac{3}{2}\nu(\nu-1) \times \frac{3}{2}\nu(\nu-1)$ identity matrix. Using this matrix allows us to encode precisely the non adjacent nodes of the graph. Notice that $(I - A)$ is a Hermitian projector as well. Thus, the same steps done for (1) can be applied.

With this results, we obtain now the following system of quadratic multivariate equation with the unknown δ to be found:

$$\begin{cases} 2 \cdot [p^T R^T A R \delta] + \delta^T R^T A R \delta = 0 \\ 2 \cdot [p^T R^T (I - A) R \delta] + \delta^T R^T (I - A) R \delta \neq 0 \end{cases} \tag{4}$$

We have now found the set of multivariate quadratic equation and inequality to describe the flexibility - hence, rigidity as well - condition of a graph. Thus, if we are able to find a vector δ which satisfies these equations, we can conclude that the graph is globally flexible. Needless to be said, finding such a δ or proving there is not one can turn out to be an extremely complicated task. The field of mathematics that deals with polynomial equations is algebraic geometry. Luckily for us there is a theorem in algebraic geometry which closely relates to our problem. This theorem gave birth to an algorithm which will help us deal with the system above. Both the theorem and the algorithm will be presented in the next section. We can now give formal proof of the linearity of the map r and that the adjacency matrix is a Hermitian projector.

C. Proofs of properties of the maps

In this subsection we will give proofs that the relative distance map r is a linear transformation and that the adjacency matrix A is a Hermitian projector. This last proof justifies the steps done in (3).

Proof 1): The relative distance map is linear

We shall prove that the map r, represented by the matrix R defined in section 2. To do so we shall prove that $r(c \cdot p + q) = c \cdot r(p) + r(q)$, with c an arbitrary real constant and p, q two position vectors of the same dimension. Without any loss of generality we can

consider the map for graphs with only three nodes. The map is the following:

$$R = \begin{bmatrix} 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}$$

Let $p = (p_1^{(x)}, p_1^{(y)}, p_1^{(z)}, p_2^{(x)}, p_2^{(y)}, p_2^{(z)}, p_3^{(x)}, p_3^{(y)}, p_3^{(z)})$ and $q = (q_1^{(x)}, q_1^{(y)}, q_1^{(z)}, q_2^{(x)}, q_2^{(y)}, q_2^{(z)}, q_3^{(x)}, q_3^{(y)}, q_3^{(z)})$. We obtain:

$$r(c \cdot p + q) = R \cdot (c \cdot p + q) = \begin{bmatrix} c \cdot p_1^{(x)} + q_1^{(x)} - c \cdot p_2^{(x)} + q_2^{(x)} \\ c \cdot p_1^{(y)} + q_1^{(y)} - c \cdot p_2^{(y)} + q_2^{(y)} \\ c \cdot p_1^{(z)} + q_1^{(z)} - c \cdot p_2^{(z)} + q_2^{(z)} \\ c \cdot p_1^{(x)} + q_1^{(x)} - c \cdot p_3^{(x)} + q_3^{(x)} \\ c \cdot p_1^{(y)} + q_1^{(y)} - c \cdot p_3^{(y)} + q_3^{(y)} \\ c \cdot p_1^{(z)} + q_1^{(z)} - c \cdot p_3^{(z)} + q_3^{(z)} \\ c \cdot p_2^{(x)} + q_2^{(x)} - c \cdot p_3^{(x)} + q_3^{(x)} \\ c \cdot p_2^{(y)} + q_2^{(y)} - c \cdot p_3^{(y)} + q_3^{(y)} \\ c \cdot p_2^{(z)} + q_2^{(z)} - c \cdot p_3^{(z)} + q_3^{(z)} \end{bmatrix}$$

which we can write as

$$c \begin{bmatrix} p_1^{(x)} - p_2^{(x)} \\ p_1^{(y)} - p_2^{(y)} \\ p_1^{(z)} - p_2^{(z)} \\ p_1^{(x)} - p_3^{(x)} \\ p_1^{(y)} - p_3^{(y)} \\ p_1^{(z)} - p_3^{(z)} \\ p_2^{(x)} - p_3^{(x)} \\ p_2^{(y)} - p_3^{(y)} \\ p_2^{(z)} - p_3^{(z)} \end{bmatrix} + \begin{bmatrix} q_1^{(x)} - q_2^{(x)} \\ q_1^{(y)} - q_2^{(y)} \\ q_1^{(z)} - q_2^{(z)} \\ q_1^{(x)} - q_3^{(x)} \\ q_1^{(y)} - q_3^{(y)} \\ q_1^{(z)} - q_3^{(z)} \\ q_2^{(x)} - q_3^{(x)} \\ q_2^{(y)} - q_3^{(y)} \\ q_2^{(z)} - q_3^{(z)} \end{bmatrix} = c \cdot R \cdot p + R \cdot q = c \cdot r(p) + r(q)$$

which concludes our proof.

Proof 2): The matrices A and $(I - A)$ define a Hermitian projector:
In order to show that the two matrices are Hermitian projectors we shall show that $A^2 = A$ and $A^T = A$. Clearly the same must hold for $(I - A)$.

Let us take a closer look to how we constructed the adjacency matrix: the matrix is 0 everywhere but in some elements in the diagonal, where it is 1. Consider a graph with nodes p_1, p_2, p_3 and let p_1p_2 and p_2p_3 be connected by an edge, respectively. The matrix is the following:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

We see that by construction A^2 will be equal to A since the diagonal elements will remain the same, having each a 0 if the diagonal element is 0 (zero row $\times$ zero column), and a 1 if the diagonal was a 1 (row$\times$column will be equal to $1^2 = 1$). All the other elements will be zero since the rows and columns in the respective row-column multiplication, even if neither were a zero vector, the 1-entries will be in different positions, resulting in a 0.

In addition to that, having only non zero entries (possibly) in the diagonal, we have that $A^T = A$ since the diagonal elements do not move by taking the transpose. With this being said, and noticing that a similar reasoning can be applied for the matrix $(I - A)$, we have proved that both matrices are Hermitian projectors.

With this proof being correct, the last step of (3) is justified and correct. In the next section we will see how to interpret system (4) as an application of the Tarski-Seidenberg theorem and how the cylindrical algebraic decomposition and the exists[] function come to our help.

III. PRACTICAL IMPLEMENTATION

The aim of this section is to give insight on how our problem relates to the fields of algebraic geometry and computer algebra, which might seem at first unrelated to our task. We will first present the Tarski-Seidenberg theorem and try to explain how it relates to our problem. Then we will look into the cylindrical algebraic decomposition and how it hypothetically solves our problem. Finally, in the last subsection we will see its practical use, explain what the limitations of the first algorithm are and provide another function that can be used for our purpose.

A. Tarski-Seidenberg Theorem

This section will present the Tarski-Seidenberg theorem[3] and explain why intuitively, the graph rigidity problem can be seen as an application of this theorem. The theorem itself is quite involved and is part of a very advanced mathematical subject: algebraic geometry.

As we found in the previous section, we have a system of multivariate quadratic equalities and inequalities that describes the rigidity conditions for a given graph. Algebraic geometry is the subject that deals with these kind of equations and explores techniques for solving them. However, these techniques (4) are out of the scope of this report and the field does not belong to my personal competences. Nevertheless, we present the following theorem and try to interpret it:

Tarski-Seidenberg Theorem: this theorem states that a set in $(n+1)$-dimensional space defined by polynomial equations and inequalities can be projected down onto n-dimensional space, and the resulting set is still definable in terms of polynomial identities and inequalities. More formally: let X be a semi-algebraic set in $\mathbb{R}^{n+1}$ and let π be the projection map that sends a set of points $(x_1, x_2, ..., x_{n+1}) \in \mathbb{R}^{n+1}$ to the set of points $(x_1, x_2, ..., x_n) \in \mathbb{R}^n$. Then $\pi(X)$ is a semi-algebraic set in $\mathbb{R}^n$.
Remark: A semi-algebraic set in $\mathbb{R}^n$ is a finite union of sets defined by a finite number of polynomial equations and inequalities.

The reader will see from the statement of the theorem that there is probably some sort of relation between it and our flexibility conditions: they are indeed a finite set of equations and inequalities in $\mathbb{R}^{3\nu}$, more specifically, one equation and one inequality. However, this

theorem by itself is not so helpful: knowing that we can map the displacement vector δ we are looking for in the flexibility conditions to vector in a smaller space does not help us that much in figuring out whether this δ exists or not. Why we presented the theorem, besides the relation with our problem, is that it gave birth to the cylindrical algebraic decomposition algorithm, which will be explained in the following subsection and will make the reader aware of why it could hypothetically be used for our purpose.

B. Cylindrical algebraic decomposition algorithm

In this subsection we will explain why the cylindrical algebraic decomposition is in theory the best option for determining an answer to our decision problem. We will also explain how to implement it in Wolfram Mathematica.
The information we will provide about the algorithm is taken by the article "How to use cylindrical algebraic decomposition" by M.Kauers. The algorithm derives from the Tarski-Seidenberg theorem we saw in the previous subsection. What it does is quantifier elimination over the reals: if we give a quantified formula it finds an equivalent formula without quantifiers. Actually it is considered to be a general tool for dealing with subsets of $\mathbb{R}^n$. If we give a description of the subset by means of polynomials equations and inequalities, it describes this set, allowing to draw different conclusions of non trivial questions. Among all, the important question it allows us to answer whether or not a given semi-algebraic set is empty, finite, open, closed, connected, or bounded.[2]

This is exactly what we need for our decision problem. As a matter of fact, for global flexibility, we do not want to know what the solutions are to (4), we are just interested in the existence or not of this solution. Thus, we want to know whether the set of solutions is empty or not. We can consider three cases for the output:

- **Empty solution set**: If the solution set is empty it means that the system has no solutions, i.e., no displacement vector can be found. Thus, we conclude that the given graph is rigid.

- **Finite solution set**: If the solution set is finite, the graph is globally but not locally flexible. To prove this, look back at definition 2 in section 2 for local rigidity: if there are finitely many δ's, then there exists one such that $\sum_{i=1}^{\nu} \delta_i$ is minimal (remark: here δ_i represents each component of the vector, not different displacement vectors). Call this displacement vector δ_{min} and let $\sum_{i=1}^{\nu} \delta_{min_i} = \epsilon_{min}$. Then we can pick a

number ϵ such that $0 < \epsilon < \epsilon_{min}$ and there would be no displacement vector such that the sum of the norm of its components is less than ϵ. Thus, the condition of definition 2 is not satisfied, meaning the graph cannot be locally flexible.

- **Infinite set of solutions**: if the set of solutions is infinite then the graph is once again globally flexible for sure. Moreover, it might also be locally flexible. The requirement for the graph to be locally flexible is that the infimum of the displacement vectors is the zero vector. This can be intuitively explained using the same reasoning we have done in the previous point: if there is an infimum which is not the zero vector then we will always be able to find an ϵ like above.

The actual implementation of the algorithm is beyond the scope of this report. Luckily we are not required to implement it: Wolfram Mathematica has a function called CylindricalDecomposition[] which takes as input the equations and inequalities together with the unknown variables and runs the algorithm by itself. The rest of the implementation is just writing down the position vectors, the two matrices and performing the operations of (4).

It almost look like we are living the perfect dream with everything that has been said so far. Unfortunately, the algorithm has doubly exponential complexity in the number of unknowns. This is very inefficient. Finding an algorithm with a smaller complexity still remains an active research field. The other bad news is that, with how we have constructed the problem, every node added, adds three unknown value (one per direction) which, with a doubly exponential complexity, increases the time by a huge amount.

We have seen that theoretically, assuming sufficient amount of time and memory in a computer, we would get a solution for the problem using the CAD algorithm. In the next subsection we will present an implementation in Wolfram Mathematica and introduce another function the program has that could help us find a solution in some specific cases.

C. Practical use and the 'Exists[]' function

In this subsection we will show the implementation in Wolfram Mathematica of one example. Later, we will show what happened after the implementation, namely, the time and the memory it took. Finally, we will see the use of another function in Mathematica which in some cases is more helpful than CAD: Exists[].

The example we will consider is a rectangle in the three dimensional space with one diagonal, where the coordinates of the nodes are $p_1 = [1\ 1\ 1]^T$, $p_2 = [3\ 1\ 1]^T$, $p_3 = [3\ 2\ 1]^T$

and $p_4 = [1\ 2\ 1]^T$. Figures (6), (7) and (8) show the Mathematica script implemented:

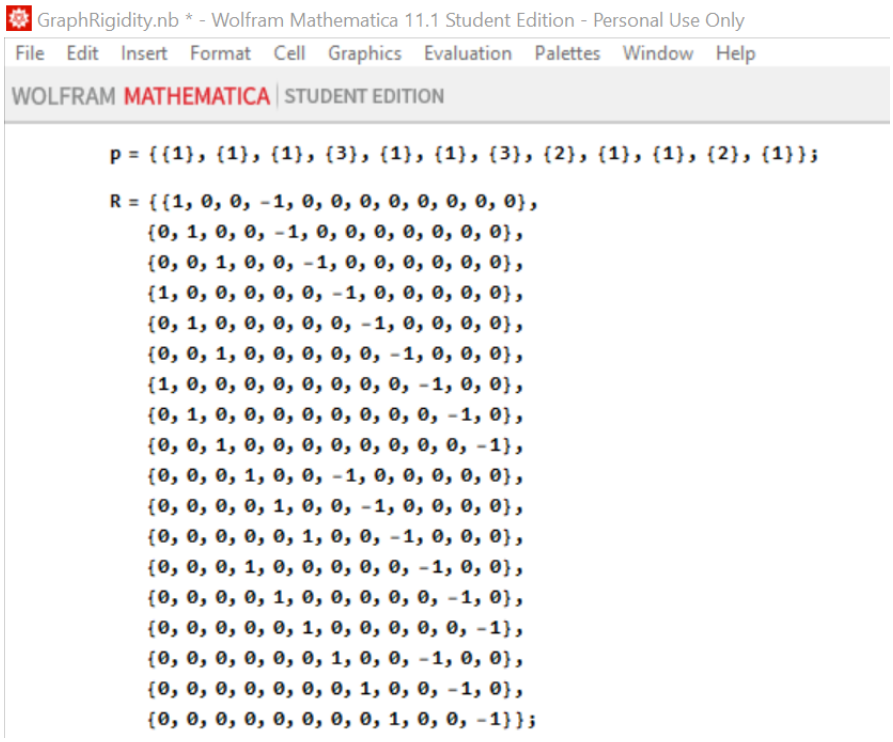

FIGURE 6: Position vector and relative distance matrix

```
A = {{1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0},
    {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1}};

d = {{d1}, {d2}, {d3}, {d4}, {d5}, {d6}, {d7}, {d8}, {d9}, {d10}, {d11}, {d12}};
```

FIGURE 7: Adjacency matrix and unknown displacement vector

```
In[93]:= factor1 = 2 * (Transpose[p].Transpose[R].A.R.d);

In[94]:= factor2 = Transpose[d].Transpose[R].A.R.d;

In[95]:= factor1 + factor2

Out[95]= {{d1 (2 d1 - d10 - d4) + d2 (-d11 + 2 d2 - d5) + d3 (-d12 + 2 d3 - d6) + d10 (-d1 + 2 d10 - d7) + d4 (-d1 + 2 d4 - d7) + d7 (-d10 - d4 + 2 d7) + d11 (2 d11 - d2 - d8) +
    d5 (-d2 + 2 d5 - d8) + 2 (-2 d1 - 2 d10 + d11 - d2 + 2 d4 - d5 + 2 d7 + d8) + d8 (-d11 - d5 + 2 d8) + d12 (2 d12 - d3 - d9) + d6 (-d3 + 2 d6 - d9) + d9 (-d12 - d6 + 2 d9)}}

In[97]:= factor3 = 2 * (Transpose[p].Transpose[R].(IdentityMatrix[18] - A).R.d);

In[98]:= factor4 = Transpose[d].Transpose[R].(IdentityMatrix[18] - A).R.d;

In[99]:= factor3 + factor4

Out[99]= {{d10 (d10 - d4) + d4 (-d10 + d4) + d11 (d11 - d5) + d5 (-d11 + d5) + d12 (d12 - d6) + d6 (-d12 + d6) +
    d1 (d1 - d7) + d7 (-d1 + d7) + d2 (d2 - d8) + d8 (-d2 + d8) + 2 (-2 d1 - 2 d10 + d11 - d2 + 2 d4 - d5 + 2 d7 + d8) + d3 (d3 - d9) + d9 (-d3 + d9)}}

CylindricalDecomposition[
 d1 (2 d1 - d10 - d4) + d2 (-d11 + 2 d2 - d5) + d3 (-d12 + 2 d3 - d6) + d10 (-d1 + 2 d10 - d7) + d4 (-d1 + 2 d4 - d7) + d7 (-d10 - d4 + 2 d7) + d11 (2 d11 - d2 - d8) +
    d5 (-d2 + 2 d5 - d8) + 2 (-2 d1 - 2 d10 + d11 - d2 + 2 d4 - d5 + 2 d7 + d8) + d8 (-d11 - d5 + 2 d8) + d12 (2 d12 - d3 - d9) + d6 (-d3 + 2 d6 - d9) + d9 (-d12 - d6 + 2 d9) == 0 &&
  d10 (d10 - d4) + d4 (-d10 + d4) + d11 (d11 - d5) + d5 (-d11 + d5) + d12 (d12 - d6) + d6 (-d12 + d6) + d1 (d1 - d7) + d7 (-d1 + d7) + d2 (d2 - d8) + d8 (-d2 + d8) +
    2 (-2 d1 - 2 d10 + d11 - d2 + 2 d4 - d5 + 2 d7 + d8) + d3 (d3 - d9) + d9 (-d3 + d9) ≠ 0,
 {d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12}]
```

FIGURE 8: Calculation of the four addends in (4) and CAD implementation

The implementation seems correct. Unfortunately, the CAD algorithm works with very few unknowns, more or less 6. Therefore already a simple rectangle generates too many unknowns for the algorithm to give an output.

We have let the code run in a device with 8GB of RAM and a CPU Intel(R) Xeon(R) CPUX5650@2.67GHz. Even with such a powerful CPU, after 36 hours the program still did not give an output. In figure (9) we show the RAM occupied by the program running: You can see that 4.75GB of RAM have been occupied by the code. We interpret this as the

FIGURE 9: CPU and RAM

program locating in memory all the solutions it has found up until now, which we reckon means that what we have discussed is correct and that it should, hypothetically, work.

As we discussed before, to determine global flexibility, we are just interested in the

existence of the solution. For this purpose there is another function we can use in Mathematica: Exists[]. More than a function, using Exists[] makes a statement of what we put inside the brackets. Therefore, if we then resolve the statement, we will get a true/false output. If the output is true then we know for sure the graph is globally flexible, but we are not able to say if it is locally flexible. Later in this subsection we will give an intuitive interpretation that can be made based on the time Mathematica takes to give the output. Below we use the existence function and resolve it for the example of the images above. We expect the output to be 'True'. Figure (10) shows the implementations of the same equations in figure (8). Thanks to this output we can conclude immediately that the graph

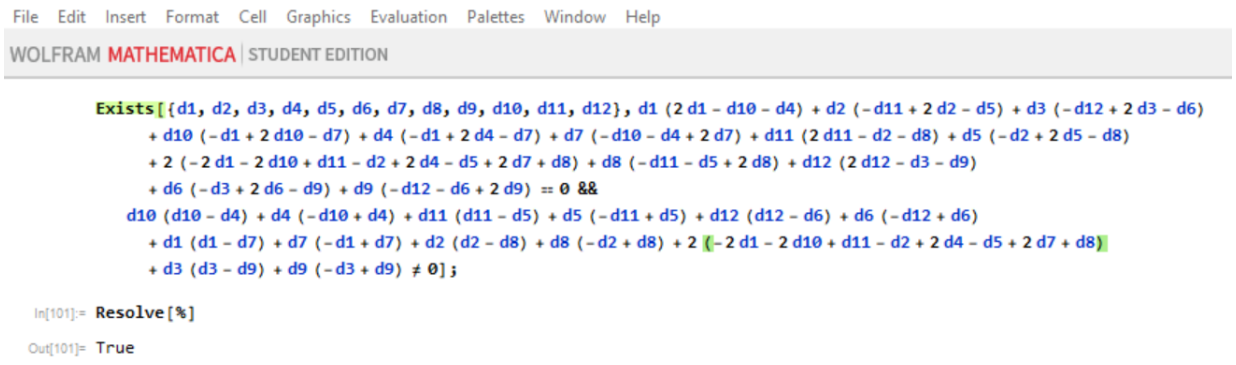

FIGURE 10: Solution existence

is globally flexible. The case in which this is practically better than CAD is when we are quite confident that the graph is flexible: this is because using Exists[] the program uses brute force and terminates as soon as it finds one solution to (4). However, if the graph is rigid, this last procedure will never terminate since it should have to test infinitely many possibilities, while CAD will, presumably, terminate.

Our thought is that, if the graph is locally flexible, like the example we just saw, the algorithm will return 'true' immediately. Else, if it takes some time to return 'true' or never terminates, then either the graph is globally flexible (locally rigid), or it's globally rigid. This is because there might be for instance only one specific solution that makes the graph flexible, and looking for it, Resolve[] in (10) might take a while finding it with brute force.

We have seen how our model can be seen as an application of the Tarski-Seidenberg theorem and how the CAD algorithm comes to use when dealing with this problem (the-

oretically, assuming we have enough time and memory in our computer). Moreover, we have seen how we can draw conclusions on the flexibility of a graph by the means of the Exists[] function in Wolfram Mathemtica.
In the next section we will give some concrete examples, from trivial to more complicated ones, and present their solutions.

IV. WORKED EXAMPLES

In this section we will provide different examples to make the theory we have explained through this report much clearer. For the examples we will draw conclusions by intuitive -still precise and correct- reasoning. Except for the examples of the first subsection, in each of the others we will always check them using Wolfram Mathematica. Each subsection will discuss a different example.

A. Complete and empty graph

In this subsection will consider two basic examples, namely a complete graph and an empty graph embedded in $\mathbb{R}^3$. We will conclude that the first one is rigid while the other is locally flexible. Before discussing the examples consider again (4):

- **Complete graph**: Let $G(\mathcal{V}, E)$ be a complete graph, i.e., $E = \mathcal{E}$. We notice that in this case the adjacency matrix A equals the identity matrix, since all possible pairs of nodes are connected by an edge. Therefore the inequality of system (4) will yield $0 \neq 0$ (since $I - A = 0$), which is a contradiction. Thus, we can conclude that there are no solutions to the system, meaning that the graph is rigid.

- **Empty graph**: Let now $G(\mathcal{V}, E)$ be an empty graph, i.e., $E = \emptyset$. Intuitively it is clear that we can move any point just a little bit and its distance from the connected nodes will remain the same, since there are no connected nodes, while its distance from the non adjacent point will change. Formally speaking, considering (4) we will have the adjacency matrix $A = 0$, thus the first equation will be $0 = 0$ which is always true. The inequality will become $2 \cdot [p^T R^T R\delta + \delta^T R^T R\delta] \neq 0$, and there are infinitely many solutions to this. Moreover the solutions can be picked

arbitrarily small. Therefore the empty graph is locally flexible.

We have shown the rigidity/flexibility properties of these two graphs. We proceed with another example.

B. Simple example: a square

In this subsection we will decide the rigidity of a square embedded in $\mathbb{R}^3$. Being a square a special case of a rectangle, we expect the same result we had for the rectangle in the introduction, namely, flexibility.

Consider a square embedded in the three dimensional eucledian space. For simplicity we will show a 2d figure but you should imagine it lying in three dimensions. As you

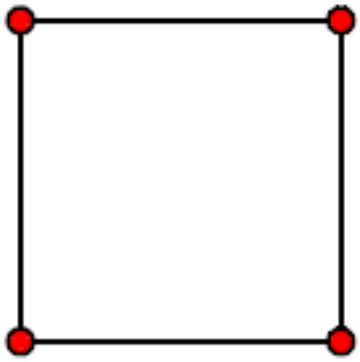

FIGURE 11: A non rigid graph

can see from figure (11), the figure respects the definition of global flexibility, i.e., we can move the top nodes to the right and downwards and we will have preserved the adjacent nodes distance but the diagonals' distances have changed, as in figure (12). To be even more precise, we realize that this 'move' can be arbitrarily small, meaning that the graph is locally flexible. We will now try to use the theory explained previously to draw the same conclusions.

First, we start by defining position vectors for the nodes. Let us run counterclockwise direction in counting the nodes starting from the bottom left one. We set the following position vectors: $p_1 = [0\ 0\ 0]^T$, $p_2 = [1\ 0\ 0]^T$, $p_3 = [1\ 0\ 1]^T$ and $p_4 = [0\ 0\ 1]^T$. Note that

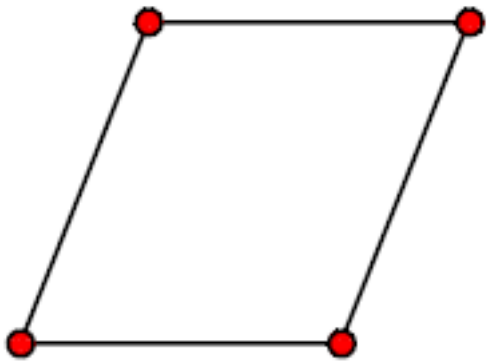

FIGURE 12: A non rigid graph

the position vectors also correspond to the coordinates of the points in the space. We can write the matrix for the map r which will look as follows:

$$R = \begin{bmatrix} I_3 & -I_3 & \mathbf{0} & \mathbf{0} \\ I_3 & \mathbf{0} & -I_3 & \mathbf{0} \\ I_3 & \mathbf{0} & \mathbf{0} & -I_3 \\ \mathbf{0} & I_3 & -I_3 & \mathbf{0} \\ \mathbf{0} & I_3 & \mathbf{0} & -I_3 \\ \mathbf{0} & \mathbf{0} & I_3 & -I_3 \end{bmatrix}$$

We can interpret the columns as being representative of p_1, p_2, p_3, p_4 and the rows of representing three times each (one per direction), $(p_1 - p_2)$, $(p_1 - p_3)$, $(p_1 - p_4)$, $(p_2 - p_3)$, $(p_2 - p_4)$, $(p_3 - p_4)$, respectively. Moreover we have the following adjacency matrix:

$$A = \begin{bmatrix} I_3 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I_3 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & I_3 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & I_3 \end{bmatrix}$$

Here the rows (and columns) can be interpreted as representative of coupling of nodes (once again, three times per couple) p_1p_2, p_1p_3, p_1p_4, p_2p_3, p_2p_4, p_3p_4, respectively. We will have the identity matrix in the entry if the nodes are connected and a 0 if they are not.

We now have defined everything needed for writing down and calculating the equation and inequality derived before in the unknown vector δ, which will be a 12×1 vector. There is no efficient way to determine a solution for the system (4), however as we can see from the figures, any circular movement of two adjacent edges will work. Without any loss of generality we decide to move vertices 3 and 4. Say we want to move them from a right angle to a 60° with respect to the 1-2 edge. This translates in our case of moving vertices 3 and 4 of $sin(\frac{\pi}{3})$ units to the left and $cos(\frac{\pi}{3})$ units downwards (remark:$60° = \frac{\pi}{6}$ rad), as shown in (11) and (12). Thus the displacement vector for this move is $\delta = (0, 0, 0, 0, 0, 0, sin(\frac{\pi}{3}), 0, -cos(\frac{\pi}{3}), sin(\frac{\pi}{3}), 0, -cos(\frac{\pi}{3}))$. This δ satisfies (4), meaning the graph is globally flexible. Moreover, we can see that the displacement vector can be taken arbitrarily small, as soon as we push a little bit one node, horizontally or vertically, also the adjacent node in that direction will move, and the diagonal length will change as well. In figure (13) the check for flexibility in Wolfram Mathematica: As expected, the output

```
In[102]:= p = {{0}, {0}, {0}, {1}, {0}, {0}, {1}, {0}, {1}, {0}, {0}, {1}};

In[103]:= 2 * (Transpose[p].Transpose[R].A.R.d) + Transpose[d].Transpose[R].A.R.d
Out[103]= {{d1 (2 d1 - d10 - d4) + d2 (-d11 + 2 d2 - d5) + d3 (-d12 + 2 d3 - d6) + d10 (-d1 + 2 d10 - d7) + d4 (-d1 + 2 d4 - d7) + d7 (-d10 - d4 + 2 d7) + d11 (2 d11 - d2 - d8) +
    d5 (-d2 + 2 d5 - d8) + d8 (-d11 - d5 + 2 d8) + d12 (2 d12 - d3 - d9) + d6 (-d3 + 2 d6 - d9) + 2 (-d1 - d10 + d12 - d3 + d4 - d6 + d7 + d9) + d9 (-d12 - d6 + 2 d9)}}

In[104]:= 2 * (Transpose[p].Transpose[R].(IdentityMatrix[18] - A).R.d) + Transpose[d].Transpose[R].(IdentityMatrix[18] - A).R.d
Out[104]= {{d10 (d10 - d4) + d4 (-d10 + d4) + d11 (d11 - d5) + d5 (-d11 + d5) + d12 (d12 - d6) + d6 (-d12 + d6) +
    d1 (d1 - d7) + d7 (-d1 + d7) + d2 (d2 - d8) + d8 (-d2 + d8) + d3 (d3 - d9) + d9 (-d3 + d9) + 2 (-d1 - d10 + d12 - d3 + d4 - d6 + d7 + d9)}}

In[105]:= Exists[{d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12},
    d1 (2 d1 - d10 - d4) + d2 (-d11 + 2 d2 - d5) + d3 (-d12 + 2 d3 - d6) + d10 (-d1 + 2 d10 - d7) + d4 (-d1 + 2 d4 - d7) + d7 (-d10 - d4 + 2 d7) + d11 (2 d11 - d2 - d8) +
      d5 (-d2 + 2 d5 - d8) + d8 (-d11 - d5 + 2 d8) + d12 (2 d12 - d3 - d9) + d6 (-d3 + 2 d6 - d9) + 2 (-d1 - d10 + d12 - d3 + d4 - d6 + d7 + d9) + d9 (-d12 - d6 + 2 d9) == 0 &&
    d10 (d10 - d4) + d4 (-d10 + d4) + d11 (d11 - d5) + d5 (-d11 + d5) + d12 (d12 - d6) + d6 (-d12 + d6) + d1 (d1 - d7) + d7 (-d1 + d7) + d2 (d2 - d8) + d8 (-d2 + d8) +
      d3 (d3 - d9) + d9 (-d3 + d9) + 2 (-d1 - d10 + d12 - d3 + d4 - d6 + d7 + d9) ≠ 0];

In[106]:= Resolve[%]
Out[106]= True
```

FIGURE 13: The square

of the Exists[] function returns true. In the next example we will see a similar example and compare it to what has been said before.

C. A rectangle with a diagonal: the difference between two dimensions and three

In this subsection we will analyze a rectangle with a diagonal. We will find out, first intuitively and then with the Wolfram Mathematica check, that the graph is flexible. This will seem contradictory to what we have said in the introduction, where it sufficed to put the diagonal in the rectangle to make it rigidm (locally rigid, to be precise). But keep in mind that in the introduction we explained rigidity giving a two dimensional example, in

this case we are lying in the three dimensional space, which gives the rectangle an additional degree of freedom to move. As a matter of fact, we aren't able to push the nodes as in the previous example, since the diagonal prevents the movement to be possible. We can, however, rotate two sides of the rectangle in three dimensions with respect to the diagonal. This idea is shown in figures (14) and (15).

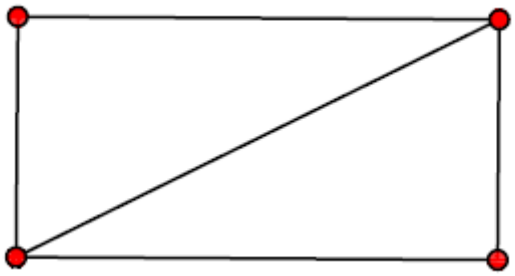

FIGURE 14: The rectangle with a diagonal

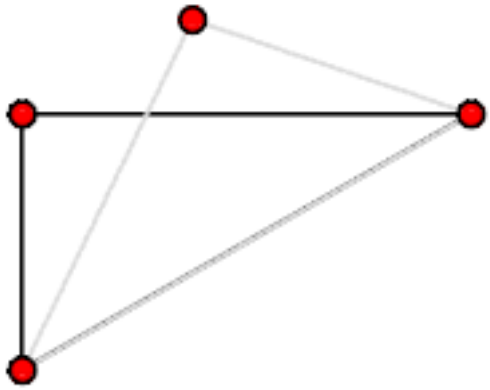

FIGURE 15: The rectangle flipped

Let the rectangle have the following position vectors: $p_1 = [1\ 1\ 1]^T$, $p_2 = [3\ 1\ 1]^T$, $p_3 = [3\ 2\ 1]^T$ and $p_4 = [1\ 2\ 1]^T$. Let also the edges connect p_1p_2, p_2p_3, p_3p_4, p_1p_3 and p_1p_4. Without any loss of generality let us rotate node 2 with respect to the diagonal of 180°. This rotation corresponds to the displacement vector $\delta = (0, 0, 0, -\frac{1}{\sqrt{5}}, 0, \frac{2}{\sqrt{5}}, 0, 0, 0, 0, 0, 0)^T$, which satisfies (4). We have found one displacement vector that works, but notice that once again we can rotate it along the diagonal of an infinitesimally small degree. This infinitesimal rotation will change the distance between nodes 2 and 4. Thus, the graph is locally flexible. We see in this case the difference between being in two dimensions

and three: while in the former the rectangle with the diagonal was globally flexible (locally rigid), in the latter it is locally flexible. As in the previous example we check with Wolfram Mathematica this case as well. Notice that, except for p, what changes from the previous example is the adjacency matrix, which is the following:

$$A = \begin{bmatrix} I_3 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & I_3 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & I_3 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & I_3 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & I_3 \end{bmatrix}$$

Of course also the addends in (4) change, as we see in figure (16).

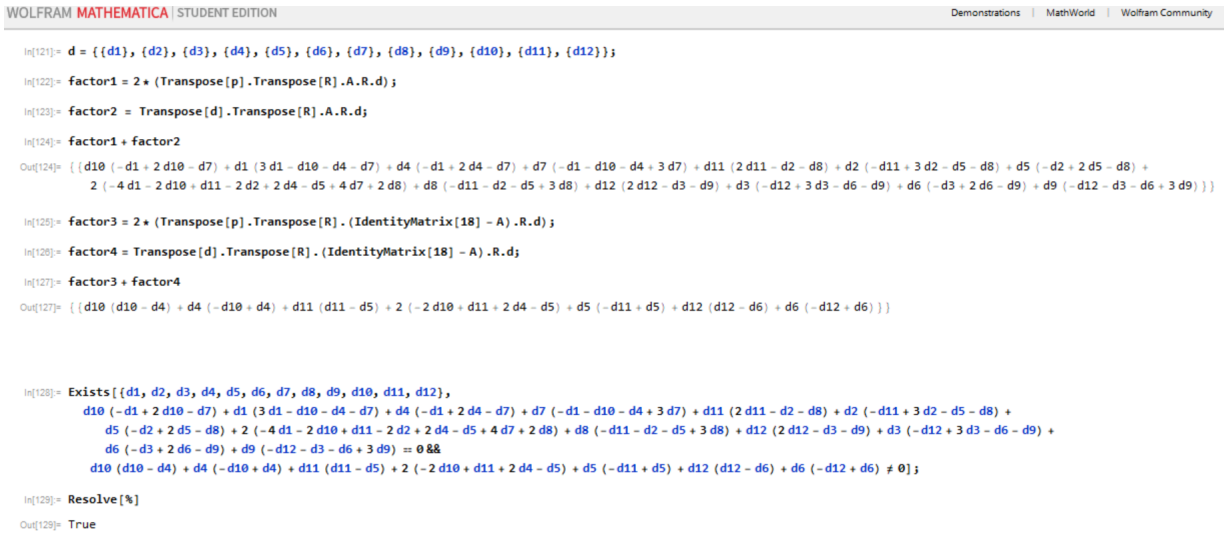

FIGURE 16: Rectangle with the diagonal

The result is once again the one expected. We will proceed with the example of a solid.

D. A three dimensional object

In this subsection we will consider two pyramids lying in two opposite sides with the same base, a square with one diagonal and having different heights, like we see in figure (19). We require the base to have the diagonal to avoid the possibility of moving the base of the solid like in example 2 of this section. It might seem like the solid we are presenting is rigid. We will first implement it in Mathematica, see the output and try to interpret

the result. We give the following position vectors for the solid: $p_1 = [0\,0\,0]^T$, $p_2 = [2\,0\,0]^T$, $p_3 = [2\,0\,2]^T$ $p_4 = [0\,0\,2]^T, p_5 = [1,\ 1,\ 4]^T$ and $p_6 = [1,\ 1,\ -2]$. We implement this in Mathematica. Figures (17) and (18) show the equation and inequality in the 18 unknowns obtained from (4) in this case and the Exists[] with its output. It is not so obvious that

```
In[136]:= factor1 = 2 * (Transpose[p].Transpose[R].A.R.d);

In[137]:= factor2 = Transpose[d].Transpose[R].A.R.d;

In[138]:= factor1 + factor2

Out[138]= {{d10 (-d1 + 4 d10 - d13 - d16 - d7) + d13 (-d1 - d10 + 4 d13 - d4 - d7) + d1 (5 d1 - d10 - d13 - d16 - d4 - d7) + d16 (-d1 - d10 + 4 d16 - d4 - d7) + d4 (-d1 - d13 - d16 + 4 d4 - d7) +
    d7 (-d1 - d10 - d13 - d16 - d4 + 5 d7) + d11 (4 d11 - d14 - d17 - d2 - d8) + d14 (-d11 + 4 d14 - d2 - d5 - d8) + d17 (-d11 + 4 d17 - d2 - d5 - d8) +
    d2 (-d11 - d14 - d17 + 5 d2 - d5 - d8) + d5 (-d14 - d17 - d2 + 4 d5 - d8) + d8 (-d11 - d14 - d17 - d2 - d5 + 5 d8) + d12 (4 d12 - d15 - d18 - d3 - d9) +
    d15 (-d12 + 4 d15 - d3 - d6 - d9) + d18 (-d12 + 4 d18 - d3 - d6 - d9) + d3 (-d12 - d15 - d18 + 5 d3 - d6 - d9) + d6 (-d15 - d18 - d3 + 4 d6 - d9) +
    d9 (-d12 - d15 - d18 - d3 - d6 + 5 d9) + 2 (-6 d1 - 4 d10 - 2 d11 + 4 d12 + 4 d14 + 12 d15 + 4 d17 - 12 d18 - 2 d2 - 6 d3 + 4 d4 - 2 d5 - 4 d6 + 6 d7 - 2 d8 + 6 d9)}}

In[140]:= factor3 = 2 * (Transpose[p].Transpose[R].(IdentityMatrix[45] - A).R.d);

In[141]:= factor4 = Transpose[d].Transpose[R].(IdentityMatrix[45] - A).R.d;

In[142]:= factor3 + factor4

Out[142]= {{d13 (d13 - d16) + d16 (-d13 + d16) + d14 (d14 - d17) + d17 (-d14 + d17) + d15 (d15 - d18) + d18 (-d15 + d18) +
    d10 (d10 - d4) + d4 (-d10 + d4) + d11 (d11 - d5) + d5 (-d11 + d5) + 2 (-2 d10 + 2 d12 + 6 d15 - 6 d18 + 2 d4 - 2 d6) + d12 (d12 - d6) + d6 (-d12 + d6)}}
```

FIGURE 17: The equations obtained

```
In[143]:= Exists[{d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12, d13, d14, d15, d16, d17, d18},
    d10 (-d1 + 4 d10 - d13 - d16 - d7) + d13 (-d1 - d10 + 4 d13 - d4 - d7) + d1 (5 d1 - d10 - d13 - d16 - d4 - d7) + d16 (-d1 - d10 + 4 d16 - d4 - d7) +
      d4 (-d1 - d13 - d16 + 4 d4 - d7) + d7 (-d1 - d10 - d13 - d16 - d4 + 5 d7) + d11 (4 d11 - d14 - d17 - d2 - d8) + d14 (-d11 + 4 d14 - d2 - d5 - d8) +
      d17 (-d11 + 4 d17 - d2 - d5 - d8) + d2 (-d11 - d14 - d17 + 5 d2 - d5 - d8) + d5 (-d14 - d17 - d2 + 4 d5 - d8) + d8 (-d11 - d14 - d17 - d2 - d5 + 5 d8) +
      d12 (4 d12 - d15 - d18 - d3 - d9) + d15 (-d12 + 4 d15 - d3 - d6 - d9) + d18 (-d12 + 4 d18 - d3 - d6 - d9) + d3 (-d12 - d15 - d18 + 5 d3 - d6 - d9) +
      d6 (-d15 - d18 - d3 + 4 d6 - d9) + d9 (-d12 - d15 - d18 - d3 - d6 + 5 d9) +
      2 (-6 d1 - 4 d10 - 2 d11 + 4 d12 + 4 d14 + 12 d15 + 4 d17 - 12 d18 - 2 d2 - 6 d3 + 4 d4 - 2 d5 - 4 d6 + 6 d7 - 2 d8 + 6 d9) == 0 &&
    d13 (d13 - d16) + d16 (-d13 + d16) + d14 (d14 - d17) + d17 (-d14 + d17) + d15 (d15 - d18) + d18 (-d15 + d18) + d10 (d10 - d4) + d4 (-d10 + d4) + d11 (d11 - d5) +
      d5 (-d11 + d5) + 2 (-2 d10 + 2 d12 + 6 d15 - 6 d18 + 2 d4 - 2 d6) + d12 (d12 - d6) + d6 (-d12 + d6) ≠ 0];

In[145]:= Resolve[%]

Out[145]= True
```

FIGURE 18: The output

the graph we chose is flexible, but it is actually globally but not locally flexible. The only allowed displacement of the graph we chose are the ones that take each top of the pyramid and reflect it with respect to its base, namely $\delta = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,4)^T$ and $\delta = (0,0,0,0,0,0,0,0,0,0,0,0,0,0,-8,0,0,0)^T$. Notice from this example that global flexibility is equivalent to local rigidity: we cannot find a continuous move that preserves the adjacent distances and changes the non adjacent one.

In the next subsection we will explain the types of examples where it becomes too hard to implement the model.

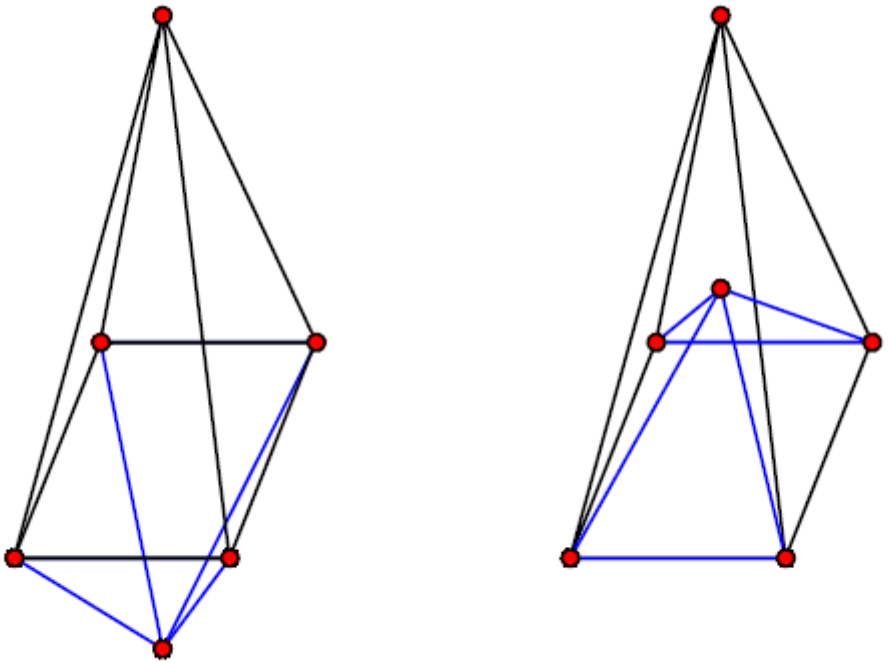

FIGURE 19: The pyramid

E. A no longer practically workable example

In this subsection we will explain the sort of examples are too big to be implemented. We have seen before the limits of the CAD algorithm and of the Exists[] function: the CAD algorithm supports in a reasonable amount of time up to 6 or 7 variables in most cases, thus, already three nodes become probably too many; on the other hand, the Exists[] function might never end should the expected output be 'false'.

One other problem with our model is the dimensions of the matrix: both R and A grow quadratically in the number of nodes. Therefore, it becomes infeasible to write down the matrices if there are too many nodes: already in the last example the adjacency matrix was 45×45 and the relative distance matrix 18×45. Both were implemented by hand. In the next section we will draw conclusions and discuss the results, we will discuss the limits of the theory developed, provide possible improvements that can be made.

V. CONCLUSIONS

In this report we have presented the problem of graph rigidity in three dimensions. By defining the right map we managed to rewrite the flexibility conditions as a system of a multivariate quadratic equation and inequality in an unknown displacement vector. Using Wolfram Mathematica we can draw conclusions on the flexibility of a given graph. If a solution to the system exists then the graph is either surely globally flexible (equivalently locally rigid) and it might be locally flexible. In theory, the CAD algorithm (provided sufficient time and memory) allows us to determine whether a graph is locally flexible: for this we would need to know that the solution set of the system is infinite and has the zero vector as an infimum.

The results obtained in this research provide now an interesting way of determining whether a graph is flexible. This, as we said in the introduction, might be of interest to the fields of engineering and maybe other natural sciences that more indirectly deal with graphs, such as chemistry or biology.

The limits of the theory is that the CAD algorithm is really inefficient, therefore, even if it works in theory, it hardly does in practice. There is a specific function used for determining the existence of solution given the equation and inequality. But even this one might fail if the graph is rigid. There are several improvements that might be made for this problem: for instance, deeper linear algebraic analysis can be made to the map introduced to possibly determine rigidity of a graph without the need of finding solutions to the system. Secondly, by a close look to many different examples, a pattern might be recognized in the coupling of the variables in the (in)equations. Moreover, an algebraic geometrical and topological approach to the problem might make several properties or solutions to special kinds of graphs arise. These are only few of the improvement points.

Finally, I would like to give one personal interesting thought about the problem regarding the implementation in Mathematica: although I have not been able to prove this, it is in my belief that, if the computer stores in memory the solution found by the Exists[] function that returns 'True', then by taking that solution and changing it by an infinitesimal amount - with certain constraints which might differ from one case to another- then it would mean that, provided that the changed solution is still a solution, the graph is locally flexible. This is because it means, in my point of view, the map that has moved the original graph to the new one is continuous.

VI. TABLE OF SYMBOLS

Symbol	Description
$G(\mathcal{V}, E)$	Graph defined by the couple of node and edge set
ϕ	Incidence function
$\mathcal{E}$	Set of all possible edges of a graph
E	Set of actual edges of a graph
$\mathcal{V}$	Set of nodes
ν	Cardinality of $\mathcal{V}$, i.e., number of nodes
p_k	Position vector of node k
e_{ij}	Edge connecting node i and j
p	Position vector of the whole graph
δ	Displacement vector
r	Relative distance map
R	Relative distance matrix
A	Adjacency matrix
I	Identity matrix

References

[1] Bill Jackson. "Notes on the Rigidity of Graphs". In: (Jan. 2002).

[2] Manuel Kauers. "How to use cylindrical algebraic decomposition". In: *Séminaire Lotharingien de Combinatoire [electronic only]* 65 (Jan. 2010).

[3] *Tarski-Seidenberg theorem.* `https://ncatlab.org/nlab/show/Tarski-Seidenberg+theorem`. Accessed on 24-06-2020.

Acknowledgments

This report was written as part of my Bachelor Thesis in which, together with my supervisor, I decided to investigate an unsolved mathematical problem that relates to one of my favorite subjects I studied during my bachelors, namely, graph theory.
There are many people that made this goal of mine possible. Before everyone else, I would like to thank my parents for always supporting me during my study and for the opportunity. Secondly, I would like all the teachers and staff of the Applied Mathematics Department for being available anytime and for teaching me everything I have learned, both academically and socially. Thirdly, I would like to thank my sister for helping me with the last minute spelling check and my uncle for the helping me with some of the figures. Then, I would like to thank my fellow students, it has been an honour fighting this battle with you, conrads. Last, but not least, a huge thank you to my 'italians in the Netherlands' friends: Federico, my roommate and big brother from whom I have learned so much during these three years; Sara, who prepared the most delicious and healthy meals ever and always being up for party; Luigi and Iris, my never-on-time tennis buddy and my spanish teacher; Francesco, the first person I met at University who always let me sleep over; Guido and Michela, for letting me play with my young 'nephew' and much more; Davide, who undoubtedly made me improve my formal english skills with great email exchanges; Federico O., for allowing me to explore the Univeristy campus in a unique way, I will be seeing you in London; Raffaele, for simply being Raffo; and Gianmarco, as we say in Italy 'breve ma intenso'. I love you all.