Università
Ca' Foscari
Venezia

Master's Degree programme — Second Cycle
(D.M. 270/2004)
in Informatica — Computer Science

Final Thesis

—

Ca' Foscari
Dorsoduro 3246
30123 Venezia

Revealing Structure in Graphs Using Regular Partitions

Supervisor
Ch. Prof. Marcello Pelillo

Candidate
Ismail Elezi
Matriculation number 848027

Academic Year
2014/2015

Ca' Foscari, University of Venice

Revealing Structure in Graphs Using Regular Partitions

by

Ismail Elezi, 848027

A thesis submitted in partial fulfillment for the degree of Master of Science

in the

Department of Environmental Sciences, Informatics and Statistics
Computer Science

February 2016

"No one knows what the right algorithm is, but it gives us hope that if we can discover some crude approximation of whatever this algorithm is and implement it on a computer, that can help us make a lot of progress."

Andrew Ng

Ca' Foscari, University of Venice

Abstract

Department of Environmental Sciences, Informatics and Statistics
Computer Science

Master of Science

by Ismail Elezi, 848027

While originally introduced as a tool in proving a long-standing conjecture on arithmetic progressions, Szemeredi's regularity lemma has emerged over time as a fundamental tool in different branches of discrete mathematics and theoretical computer science. Roughly, it states that every graph can be approximated by the union of a small number of random-like bipartite graphs called regular pairs. In other words, the result provides us a way to obtain a good description of a large graph using a small amount of data, and can be regarded as a manifestation of the all-pervading dichotomy between structure and randomness.

However, the non-constructive nature of the lemma made its usefulness limited only in theoretical mathematics and computer science for around two decades. In the nineties, things changed when two different algorithmic versions of the lemma were developed, and by the end of last decade, the lemma was finally used in practice.

This thesis is a tentative to study the regularity lemma in context of structural pattern recognition. We will use the regularity lemma to compress some graph, and then study the reduced graph, knowing that it inherits the main properties of the original graph. By doing so, we will save both computer memory and CPU time.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, professor Marcello Pelillo who guided my work from the very beginning, who was happy to meet me every time I was getting stuck in my work, and with whom we had many fruitful discussion which generated ideas that made their way into the thesis. Furthermore, his awesome lectures in the field of Artificial Intelligence and Information Theory is a reason why I am planning to get heavily involved in the field of Artificial Intelligence in general, and Machine Learning in particular.

I would like to mention my friends from the university who had some contribution on the work done in the thesis or during the entire course of the studies. Andrea Lazzarotto for answering my logistics questions about the deadlines and forms we had to fill, and with whom we had many discussions during our studies, especially in Machine Learning and Computer Vision courses (the coffee machine was a good mutual friend); Marco Fiorucci is doing his PhD in a closely related topic, and with him we had discussions on the high level ideas of the lemma; Eyasu Zemene Mequanint has used dominant sets and replicator dynamics in the past, and he helped me initiate with the replicator dynamics algorithm.

Last but certainly not least, I would like to thank my family for their unconditional moral and financial support during the studies, and for accommodating my weird study-sleep needs during the time I worked on the thesis. This work is as much their contribute, as it is mine. I hope that with this thesis I have made them a little bit proud! . . .

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The twenty-first century, can be easily called as the century of the big data. In an interview in 2010, the CEO of Google, Eric Schmidt said [41]: *"There was 5 exabytes of information created between the dawn of civilization through 2003, but that much information is now created every 2 days, and the pace is increasing...People aren't ready for the technology revolution that's going to happen to them."* [1] Data has been routinely called, 'the new oil'.

Saying that, the big data came with a problem: 'how to interpret them'. Because the data have become now so cheap, and because there is a lot of data right there, it has become very difficult and expensive to proceed and mine the collected data. New algorithms that can work with big data, and get results in acceptable times, are more needed than ever.

Graph theoretic approaches usage has been increased during the last two decades. In particular, the notion of similarity - which is closely related to graphs - has started gaining territory compared with the traditional feature based algorithms. However, there are still graph problems which have high complexity, and so are unable to work with the big data.

This thesis has two main points. The first one being the regularity lemma of Endre Szemerédi, a mathematical tool used in the field of extremal graph theory. The second one is the notion of graph isomorphism. The ultimate goal of the thesis is to provide a

[1] While Schmidt's numbers have been debated, and it looks that the real numbers have been slightly different, the quotes illustrate well his point, and the implication of the very big data.

platform of using the regularity lemma in order to make the process of graph matching more feasible.

1.1 Motivation

The regularity lemma of Szemerédi is one of the most important results in the field of extremal graph theory. It was initially built as a tool in order to prove the now-called Szemerédi's theorem, but soon after, it was understood that its implications are bigger, and so it can be widely used. However, the original version provided by Szemerédi was non-constructive, which limited its usage outside theoretical mathematics and graph theory. But because of its power, many researchers worked on it, and at least two constructive versions of the lemma were provided, while also the lemma was extended to work for weighted graphs, and ultimately for hypergraphs.

The lemma roughly states that every graph can be well-approximated as a union of bipartite graphs, such that: 1) most edges run between different parts; 2) those edges behave as if generated at random. Central to the lemma is the notion of regularity, which is closely related to the notion of randomness.

An important aspect of the lemma is that if we consider the new 'bipartite' graphs, as vertexes of a new graph, and the density (number of edges) between those graphs as edges, the new graph seems to inherit most of the properties of the original graph. In literature, this new graph is called as essence or substance of the original graph. For many problems, this reduced graph can be studied and then the acquired knowledge can be expanded back to the original 'much larger' graph.

It needs to be said that despite the fact that constructive versions of the lemma now exist, they are extremely expensive in memory and time. In fact, in order to get the guaranteed solution from those algorithms, graphs need to be astronomically large, larger than what can be currently stored even in supercomputers. This is a reason why the lemma is not widely used in applied computer science. But it can be seen that relaxed versions of the lemma - which work for most, but not all graphs - exist. Saying that, even in the original versions, the lemma isn't understand well, so changing it would make our theoretical knowledge about it even smaller. Still this seems to be the only way of working with the lemma. While the original heuristics were developed by Sperotto and

Pelillo [44], here at Ca' Foscari, we looked at some new heuristics which can improve the results of the lemma.

The graph isomorphism problem is the problem of determining whether two graphs are isomorphic or not, and if so, to provide an isomorphism between them. At first look, it might be seen that the problem is interesting only on pure mathematics, but graph isomorphism is extensively used in applied sciences. Some examples of usage of graph isomorphism are:

- In cryptography, more precisely, in zero-knowledge proofs.

- In chemiinformatics and computational chemistry, graph isomorphism testing is used to identify a chemical compound within a chemical database.

- In automata theory, the graph isomorphism can be used to show that two languages are equal.

- In analysis of social structure (including social networks).

- In fingerprint matching.

- In electronic design automation, graph isomorphism is the basis of the verification whether the electric circuits represented by a circuit schematic and an integrated circuit layout are the same [4], etc.

Despite it is widely used in very different fields, no polynomial algorithm has been found. In fact, this problem is one of those rare problems for which no polynomial algorithm has been found, while also there doesn't exist any proof which shows that the problem is NP-complete. At the moment, most researchers believe that the problem lies strictly between the P and NP-complete classes.

Of course, this hasn't stopped investigators working on the problem. There are some precise algorithms (completely useless for large graphs), but more importantly, there exist approximate algorithms which can solve the problem significantly faster than those precise algorithm. One of the possible ways of doing it is to transform the problem of graph isomorphism into the problem of maximal clique and solving that problem, and it has been shown that the corresponding maximal clique can be seen as an isomorphism between the graphs. Maximal clique problem is even harder to be solved in precise manner (proven to be NP-hard), so to solve it we need to use some heuristics, which solve the problem in an inexact way. A possible way of doing so, is by using replicator dynamics, a class of equation developed in evolutionary game theory.

The described algorithm (and other similar algorithms) don't perform well if the graphs are very large. The results might not be great, while also the time needed to solve the problem is quite long. We thought that a way of making the problem easier is to use the regularity lemma in order to reduce the graphs, then match the reduced graphs, and then finally expand those graphs and match the big graphs. This seems that not only solves the problem faster, but also makes possible to get an approximate matching on the big graphs, which isn't possible to be done in personal computers if we use directly the matching algorithm (without the compression part provided by the regularity lemma).

1.2 Structure of the Thesis

The structure of the thesis is as follows:

- Chapter 2 starts by briefly describing the field of extremal graph theory, and Szemerédi's theorem in particular. Then, we have given some definitions, like the edge density, regularity and equitability. In the end of the chapter, we show the regularity lemma and give a proof of it.

- Chapter 3 builds from the previous chapter and gives constructive versions of the lemma. We first mention the Alon algorithm (which is based on the notion of neighborhood deviation) and Frieze-Kannan algorithm (which is based on the notion of singular value decomposition). The algorithms are very similar to each other, despite that they have completely different central ideas.

Because both algorithms require astronomically large graphs, we have tried to make a new heuristic algorithm which works for 'normal-sized' graphs. The caveat is that there is no theoretical guarantee that the algorithm will converge or that the reduced graphs actually inherit the properties of the original graphs. However, the empirical evidence shows that for most graphs, the heuristic algorithm seems to work.

Finally, we end the chapter by extending the idea to weighted graphs.

- Chapter 4 starts by describing the problem of graph isomorphism. It continues by transforming it in the problem of finding a maximal clique in the association graph, by using the replicator dynamics. We show how the process works in a toy case.

We conclude the chapter by extending the results to the weighted graphs. This basically means, generalizing the maximal clique problem into the problem of finding dominant sets.

- Chapter 5 is the chapter when we put everything together, and combine the two algorithms. We show how the regularity lemma performs, and we show how the graph matching algorithm perform. Then, we give some results in how well the combined algorithm (reduce the graphs - match the small graphs - expend the results to the large graphs) performs. We use both real and synthetic datasets.

- Chapter 6 is the final chapter on the thesis. We review the work done in the previous chapters, make some conclusions and mention what can be done in future work.

We hope that by the end of the thesis, at least we have got a better idea on the usefulness of the regularity lemma in the process of graph isomorphism and graph matching.

Chapter 2

Szemerédi's Regularity Lemma

2.1 Extremal Graph Theory

Extremal graph theory is a branch of the mathematical field of graph theory, which originated in the middle of twentieth century from the work of Pal Turan and Paul Erdos and was studied later by a lot of researchers, with probably the most famous ones being Bela Bollobas, Endre Szemerédi, Arthur Stone and Reinhard Deistel.

As written in Diestel book [16] extremal graph theory deals with questions like: How many edges, for instance, do we have to give a graph on n vertices to be sure that, no matter how these edges happen to be arranged, the graph will contain a K^r subgraph for some given r? Or at least a K^r minor? Or a topological K^r minor? Will some sufficiently high average degree or chromatic number ensure that one of these substructures occurs?

In essence, it can be said that extremal graph theory studies the graph invariants and studies how the structure of graphs ensures certain properties of graphs (e.g. cliques, coloring) under specific conditions (e.g., edge density and minimum degree).

A graph invariant is a function which takes graphs as inputs and assigns equal values to isomorphic graphs. The graph invariants can range from simple ones (like the number of edges, the number of vertices) to more complex ones (like minimum/maximum degree, chromatic number or diameter).

In Diestel book [16] , extremal problems are divided into two categories:

- Looking for ways to ensure by global assumptions that a graph G contains some given graph H as a minor.

- In the second category there are all the problems in which we are asking what global assumptions might imply the existence of a given graph H as subgraph.

The regularity lemma of Szemerédi, is one of the most important tools to demonstrate most of the extremal existence theorems. Not surprisingly, for the majority of its existence, it had been used purely for these type of problems.

2.2 Szemerédi's Theorem

The regularity lemma of Szemerédi [46] is perhaps the most central result in extremal graph theory. Before introducing the reader to the lemma, I would like to quote professor Luca Trevisan from his blog [49] : *"Szemerédi's theorem on arithmetic progressions is one of the great triumphs of the 'Hungarian approach' to mathematics: pose very difficult problems, and let deep results, connections between different areas of math, and applications, come out as byproducts of the search for a solution".*

In some way in order to deeply understand the power of it, you have to go back to Erdos and Turan conjecture [17] on Ramsey's properties of arithmetic progression.

Theorem 2.1 (Erdos and Stone 1946) - *For all integers $r \geq 2$ and $s \geq 1$, and every $\epsilon > 0$, there exists a n_0 such that every graph with $n \geq n_0$ vertices and at least*

$$t_{r-1}(n) + \epsilon n^2 \tag{2.1}$$

edges contains K_s^r as a subgraph.

While the proof of the theorem is beyond the scope of this thesis, it needs to be said that Szemerédi was the first one to prove it in [45] , and while doing it, the regularity lemma served as a powerful tool on proving the main theorem. Furthermore, Szemerédi got awarded the Abel Price in Mathematics (2012) partially for the work done in the theorem, while two field medals (Timothy Gowers, 1998 and Terence Tao, 2006) were awarded at least partially for their results on the regularity lemma, and using it to prove fundamental results in arithmetic progressions.

On this section we are going to introduce the reader to the regularity lemma and its proof. It needs to be said that the lemma on its original form is non-constructive, so in the following chapter we are going to build on the idea and provide two constructive versions of the lemma.

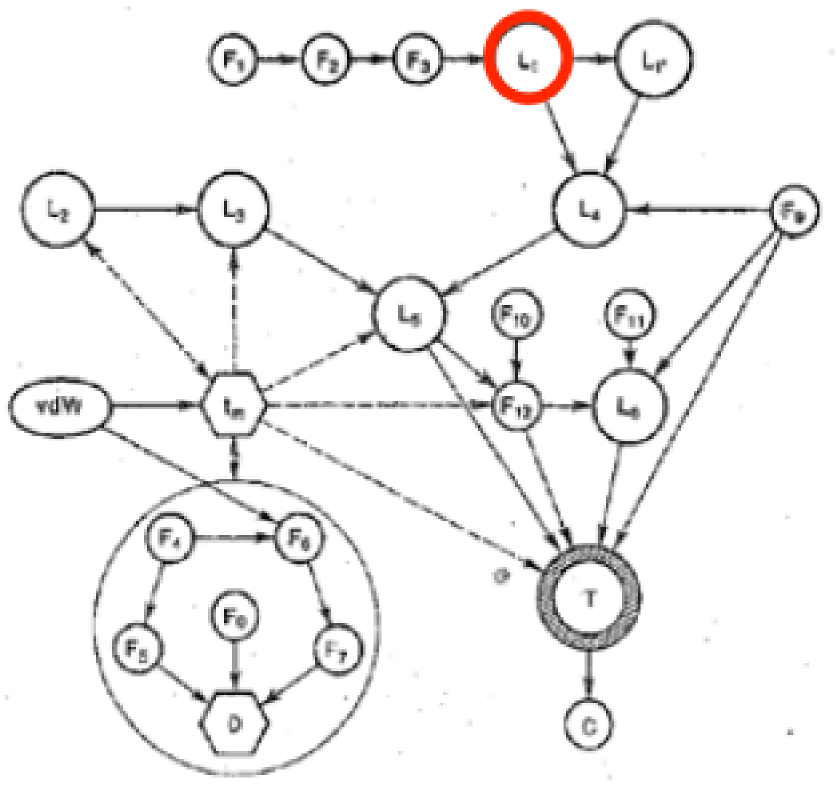


FIGURE 2.1: While proving the Erdos-Turan conjecture (T in the middle), Szemerédi needed to build the infrastructure for it. The regularity lemma, is called L1 and is circled on red [45].

2.3 The Regularity Lemma

Before we will give the lemma, we are going to provide a few definitions as given in [46], [14], [44], [1], [18] [40].

2.3.1 Some definitions

Definition 2.2 - *A simple graph is a pair $G = (V, E)$, where V is a finite set, called the vertices of G, and E is a subset of the set of two-element subsets of V, called the edges of V.*

Definition 2.3 - *Let $G = (V, E)$ be an undirected graph with no self-loops, where V is the set of vertices and E is the set of edges, and let $X, Y \subseteq V$ be two disjoint subsets of vertices of G. We define the edge density of the pair (X,Y) as:*

$$d(X, Y) = \frac{e(X, Y)}{|X|\,|Y|} \tag{2.2}$$

where e(X,Y) denotes the number of edges of G with an endpoint in X and an endpoint in Y, and $|\cdot|$ denotes the cardinality of a set. The edge densities are real numbers between 0 and 1.

Definition 2.4 - Given a positive constant $\epsilon > 0$, we say that the pair (A,B) of disjoint vertex sets $A, B \subseteq V$ is ϵ-regular if for every $X \subseteq A$ and $Y \subseteq B$ satisfying

$$|X| > \epsilon\,|A| \quad \textit{and} \quad |Y| > \epsilon\,|B|$$

we have

$$|d(X, Y) - d(A, B)| < \epsilon \tag{2.3}$$

Informally, this means that in an ϵ-regular bipartite graph, the edge density between any two sufficiently (depending on ϵ) large subsets is about the same as the original density. This implies that all the edges are distributed almost uniformly, giving the perception that the edges are random. It can be easily seen, that in order to satisfy the above equation for every ϵ, the graph should be a complete bipartite graph.

Definition 2.5 - *A partition P of the vertex set $V = V_0 \cup V_1 \cup \ldots \cup V_k$ of a graph $G = (V, E)$ is called an equitable partition if all the classes $V_i, 1 \leq i \leq k$ have the same cardinality. V_0 is called the exceptional class. The exceptional class V_0 (which may be empty) serves only a technical purpose: it makes possible that all the other classes have exactly the same number of vertices.*

Definition 2.6 - *An equitable partition P of the vertex set $V = V_0 \cup V_1 \cup \ldots \cup V_k$ of a graph $G = (V, E)$ is called ϵ-regular if $|V_0| < \epsilon\,|V|$ and all but $\epsilon\binom{k}{2}$ of the pairs (V_i, V_j) are ϵ-regular where $1 \leq i < j \leq k$.*

Definition 2.7 - *For an equitable partition P of the vertex set $V = V_0 \cup V_1 \cup \ldots \cup V_k$ of $G = (V, E)$, we associate a measure called the index (or potential) of P which is defined by:*

$$ind(P) = \frac{1}{k^2}\sum_{s=1}^{k}\sum_{t=s+1}^{k} d(C_s, C_t)^2 \tag{2.4}$$

The index will measure the progress towards an ϵ-regular partition. Clearly, it is a number bounded from 0 and 0.5. In fact

$$\lim_{k\to\infty} ind(P) = \frac{1}{2} \tag{2.5}$$

if and only if, the graph is the union of complete bipartite graphs.

The last quantity comes directly from the density of a pair being bounded by 1 (a complete bipartite graph has the highest density from all possible graphs, and that density is 1 and comes from the definition). If the density is being bounded by 1, then so it is the square of density.

Assuming that the graph G is the union of complete bipartite graphs, we get:

$$ind(P) = \frac{1}{k^2}\sum_{s=1}^{k}\sum_{t=s+1}^{k} 1 \tag{2.6}$$

Solving the double sum we get:

$$ind(P) = \frac{k(k-1)}{2k^2} = \frac{k^2-k}{2k^2} \tag{2.7}$$

and whose limit is 1/2.

2.3.2 The Lemma

Before we give the regularity lemma, we need an another lemma. The lemma was given and proven by Szemerédi himself [46].

Lemma 2.8 - *Let $G = (V, E)$ be a graph with n vertices. Let P be an equitable partition of V into classes $C_0, C_1, ..., C_k$, the exceptional class being C_0. Let ϵ be a positive integer such that*

$$4^k > 600\epsilon^{-5}$$

If more than ϵk^2 pairs (C_s, C_t) in $1 \leq s < t \leq k$ are s-irregular then there is an equitable partition Q of V into $1 + k4^4$ classes, the cardinality of the exceptional class being at most

$$|C_0| + \frac{n}{4^k}$$

and such that

$$ind(Q) > ind(P) + \frac{\epsilon^5}{20} \tag{2.8}$$

Now, we are in a position to give the regularity lemma of Szemerédi. We are giving it as it was originally given in [46].

The Regularity Lemma 2.9 - *For every positive real ϵ and for every integer m there are positive integers N and M with the following property: for every graph G with at least N vertices there is an ϵ-regular partition of G into k+1 classes such that $m \leq k \leq M$*

Proof - Let s be the smallest integer such that

$$4^s > 600\epsilon^{-5}, s > m, s \geq \frac{2}{\epsilon} \tag{2.9}$$

Define a sequence f(0), f(1), f(2),... by setting f(0) = s and

$$f(t+1) = f(t)4^{f(t)} \tag{2.10}$$

for every t. Let t be the largest non-negative integer for which there exists an equitable partition P of V into $1 + f(t)$ classes, such that

$$ind(P) \geq \frac{t\epsilon^5}{20} \tag{2.11}$$

and the size of exceptional class does not exceed

$$\epsilon n(1 - \frac{1}{2^{t+1}}) \tag{2.12}$$

Such a partition certainly exists for t $= 0$. Since $ind(P) \leq \frac{1}{2}$ for every partition P, the integer t is well defined. By our lemma, and by the maximality of t, the partition P is ϵ-regular. Hence, we may set $M = f(\lfloor 10\epsilon^{-5} \rfloor$.

2.3.3 Discussion

Stripping the mathematics, what the lemma basically says is: Given an $\epsilon > 0$, the nodes of any graph can be partitioned into a relatively small number (depended on ϵ) of equal parts (bar the exceptional class V_0), so that most bipartite graphs between two parts are essentially random (with different densities).

The lemma is non-constructive. Neither it, nor its proof give an algorithm how to build this regular partitions. In fact, in extremal graph theory (and other mathematics sub-disciplines, like Information Theory) many of the central results are given in some non-constructive format. For around two decades - while the lemma was important in graph theory - it wasn't considered in computer science. However, in the nineties there were some good developments, which gave rise to constructive versions of the lemma, and finally they resulted with algorithms which can be used in practice.

To conclude this section, we need to say that while the lemma originally was built to work for dense graphs, Kohayakawa gave a version of it which works for sparse graphs in [25]. Later, he worked on a version of regularity lemma for hyper-graphs. The readers are referred to papers [26] and [39].

Chapter 3

Constructive Versions of the Regularity Lemma

In the previous chapter, we gave an overview of Szemerédi's theorem, and then we introduced the regularity lemma. However, we mentioned that the lemma is non-constructive, so for around two decades the lemma was used only in pure mathematics. This changed in the nineties, when two different constructive versions of the lemma were made.

3.1 Alon-Duke-Lefmann-Rodl-Yuster Algorithm

The Szemerédi's paper on regularity lemma [46] ended with the author rising the following question: does the lemma hold if we change the maximum number of irregular pairs from $\epsilon\binom{k}{2}$ to 0. However, this was proven false by the work of L. Lovasz, P. Seymour, T. Trotter and the work presented in Alon et al. [1]. A simple way of showing that irregular pairs are necessary for the regularity lemma to work can be shown by taking a bipartite graph with classes $A = \{a_1, ..., a_n\}$ and $B = \{b_1, ..., b_n\}$ in which each $a_i b_j$ is an edge if and only if $i \leq j$.

The problem of checking if a given partition is ϵ-regular is quite interesting from the complexity point of view. As proven from [1], constructing an ϵ-regular partition is easier than checking if a given one is ϵ-regular or not.

Theorem 3.1 [1] - *The following decision is co-NP-complete. Given an input graph G, an integer $k \geq 1$ and a parameter $\epsilon > 0$, and a partition of the set of vertices of V into $k+1$ parts. Decide if the given partition is ϵ-regular.*

The proof of the theorem actually gives that the problem remains co-NP-complete even for $\epsilon = 1/2$ and for $k = 2$.

Theorem 3.2 (A constructive version of the Regularity Lemma [1]) - *For every $\epsilon > 0$ and every positive integer t there is an integer $Q = Q(\epsilon, t)$ such that every graph with $n > Q$ vertices has an ϵ-regular partition into $k+1$ classes, where $t \leq k \leq Q$. For every fixed $\epsilon > 0$ and $t \geq 1$ such a partition can be found in $O(M(n))$ sequential time, where $M(n)$ is the time for multiplying two n by n matrices with 0,1 entries over the integers. It can also be found in time $O(logn)$ on an EREW PRAM with a polynomial number of parallel processors.*

3.1.1 Finding a Regular Partition in Polynomial Time

This entire subsection will be devoted to proving the previous theorem. The reason for doing so, is that the algorithm is contained inside the proof, and by understanding the proof, the reader will understand the algorithm too.

The brief idea of the algorithm is that given some graph G and $\epsilon > 0$, it computes some $\epsilon' < \epsilon$ (ϵ' being a function of ϵ). In case G is not ϵ-regular, the algorithm correctly reports this and provides the evidence that the graph G is not ϵ' ($< \epsilon$)-regular. In case G is ϵ'-regular, the algorithm decides that the graph is ϵ-regular too. In all the other cases, the algorithm behaves to one of the above two possibilities and the user has no control on the possibility it chooses.

The proof of the previous theorem is quite lengthy, so it has to be seperated within a few theorems as in [1]. Before that we need a few definitions.

Definition 3.3 - *Let H be a bipartite graph with equal color classes $|A| = |B| = n$. We define the average degree of H as*

$$d = \frac{1}{2n} \sum_{i \in A \cup B} deg(i) \tag{3.1}$$

Definition 3.4 - *Let H be a bipartite graph with equal color classes $|A| = |B| = n$. Let d be the average degree of H. For two distinct vertices $y_1, y_2 \in B$ define the neighbourhood deviation of y_1 and y_2 by*

$$\sigma(y_1, y_2) = |N(y_1) \cap N(y_2)| - \frac{d^2}{n} \tag{3.2}$$

For a subset $Y \subset B$ the deviation of Y is defined

$$\sigma(Y) = \frac{\sum_{y_1, y_2 \in Y} \sigma(y_1, y_2)}{|Y|^2} \tag{3.3}$$

Lemma 3.5 [1] - *Let H be a bipartite graph with equal classes $|A| = |B| = n$, and let d denote the average degree of H. Let $0 < \epsilon < \frac{1}{16}$. If there exists $Y \subset B$, $|Y| \geq \epsilon n$ such that $\sigma(Y) \geq \frac{\epsilon^3}{2} n$ then at least one of the following occurs.*

1. *$d < \epsilon^3 n$.*

2. *There exists in B a set of more than $\frac{1}{8}\epsilon^4 n$ vertices whose degree deviate from d by at least $\epsilon^4 n$.*

3. *There are subsets $A' \subset A$, $B' \subset B$, $|A'| \geq \frac{\epsilon^4}{4} n$, $|B'| \geq \frac{\epsilon^4}{4} n$ and $|d(A', B') - d(A, B)| \geq \epsilon^4$.*

Furthermore, there is an algorithm whose input is a graph H with a set $Y \subset B$ as above that outputs either:
(i) The fact that (1) holds, or
(ii) The fact that (2) holds and a subset of more than $\frac{1}{8}\epsilon^4 n$ members of B demonstrating this fact, or
(iii) The fact that (3) holds and two subsets A' and B' as in (3) demonstrating this fact. The algorithm runs in sequential time $O(M(n))$, where $M(n) = O(n^{2.376})$ [1] [2] is the

[1]It needs to be said that the asymptotic time $O(n^{2.376})$ comes from the Coppersmith-Winograd algorithm which at the time when the paper was published, was the algorithm with the best asymptotic time. Since then, there have been a few improvements of the algorithm, with the current fastest algorithm having asymptotic time of $O(n^{2.373})$.

[2]Despite that the algorithms mentioned in the previous footnore have great asymptotic time, they aren't used in practice. The reason for that is because they work better than the standard algorithms,

time needed to multiply two n by n 0,1 matrices over the integers, and can be parallelized and implemented in time $(O(logn)$ on an EREW PRAM with a polynomial number of processors.

Proof [1] - We assume that cases (1) and (2) don't happen, and in that case we prove than (3) must happen. Denote $Y' = \{y \in Y \mid |deg(y) - d| < \epsilon^4 n\}$. Y' cannot be empty because case (2) does not happen. Choose an $y_0 \in Y$ that maximizes $\sum_{y \in Y} \sigma(y_0, y)$. Estimating this quantity:

$$\sum_{y' \in Y'} \sum_{y \in Y, y \neq y'} \sigma(y', y) = \sigma(Y) |Y|^2 - \sum_{y' \in Y \setminus Y'} \sum_{y \in Y, y \neq y'} \sigma(y', y) \geq \frac{\epsilon^3}{2} n |Y|^2 - \frac{\epsilon^4}{8} n^2 |Y| \tag{3.4}$$

$|Y'| \leq |Y|$ so:

$$\sum_{y \in Y} \sigma(y_0, y) \geq \frac{\epsilon^3}{2} n |Y| - \frac{\epsilon^4}{8} n^2 \geq \frac{3}{8} \epsilon^3 n |Y| \tag{3.5}$$

There are at least $\frac{\epsilon^4}{4} n$ vertices $y \in Y$ whose neighbourhood deviation is greater than $2\epsilon^4 n$. If there were less - using the fact that the neighbourhood deviation is upper bounded from n - we would get:

$$\sum_{y \in Y} \sigma(y_0, y) \leq \frac{\epsilon^4}{4} n^2 + |Y| 2\epsilon^4 n \leq \frac{\epsilon^3}{4} n |Y| + 2\epsilon^4 n |Y| < \frac{\epsilon^3}{8} \epsilon^3 n |Y| \tag{3.6}$$

which contradicts equation (3.5)

Set $B' \cup Y$, $|B'| \geq \frac{\epsilon^4}{4} n^2$, $y_0 \notin B'$ and for every vertex $b \in B'$ we have $|N(b) \cap N(y_0)| > \frac{d^2}{n} + 2\epsilon^4 n$. Define $A' = N(y_0)$. Then, $|A'| \geq d - \epsilon^4 n \geq \epsilon^3 n - \epsilon^4 n \geq 15\epsilon^4 n > \frac{\epsilon^4}{4} n$. Show that $|d(A', B') - d(A, B)| \geq \epsilon^4$:

only for large matrixes, matrixes which cannot be processed by modern hardware. In practice, the most used matrix multiplication algorithms are Strassen Algorithm, which has asymptotic time $O(n^{2.8})$ and BLAS subroutines which are used in different programming libraries. On the implementation, we used numpy.dot function to do the matrix multiplication. The asymptotic time is worse than the promised asymptotic time from the lemma.

$$e(A', B') = \sum_{b \in B'} |N(b) \cap N(y_0)| > \tfrac{|B'|d^2}{n} + 2\epsilon^4 n\, |B'| \tag{3.7}$$

Finally,

$$d(A', B') - d(A, B) > \frac{d^2}{n\,|A'|} + \frac{2\epsilon^4 n}{|A'|} - \frac{d}{n} \geq \frac{d^2}{n(d + \epsilon^4 n)} + 2\epsilon^4 - \frac{d}{n} = 2\epsilon^4 - \frac{d\epsilon^4}{d + \epsilon^4 n} \geq \epsilon^4. \tag{3.8}$$

This completes the proof of the non-algorithmic part of the lemma. The algorithmic part comes directly from the lemma:

1a) Check if (1) holds. That can be easily done in $O(n^2)$ time. If $d < \epsilon^3 n$ then that partition can be declared regular and you are done.

2a) Check if (2) holds. In order to do so, count the number of vertices in B whose degree deviate from d by at least $\epsilon^4 n$. If there are more than $\frac{\epsilon^4}{8} n$ such vertices, then at least half of them deviate in the same direction. Call the set of those vertices (deviating in the same direction) as B'. Clearly, $|B'| \geq \frac{\epsilon^4 n}{16}$. If this condition holds, then save B' as certificate (witness) and $B \setminus B'$ as the complement (needed later when we do the partition step). The certificate for A is A itself (complement being the empty set).
If neither of these two conditions hold, then proceed to the following step.

3a) For each $y_0 \in B$ with $|deg(y_0) - d| < \epsilon^4 n$ we find the set of vertices $B_{y0} = \{y \in B | \sigma(y_0, y) \geq 2\epsilon^4 n\}$. The non-algorithmic part of the proof guarantees the existence of at least a y_0 for which $|B_{y0}| \geq \frac{\epsilon^4 n}{4}$. The subsets $B' = B_{y0}$ and $A' = N_{y0}$ are the required certificates. The quantities $\sigma(y, y')$ for $y, y' \in B$ can be done by squaring the adjacency matrix of the pair (more precisely, multiplying the adjacency matrix with the transposed adjacency matrix).

Now we need a few other lemmas to completely proof the new version of the regularity lemma.

Lemma 3.6 [1] - *Let H be a bipartite graph with equal classes $|A| = |B| = n$. Let $2n^{-\frac{1}{4}} < \epsilon < \frac{1}{16}$.. Assume that at most $\frac{\epsilon^4 n}{8}$ vertices of B deviate from the average degree* *of H by at least $\epsilon^4 n$. Then, if H is not ϵ-regular there exists $Y \cup B$, $|Y| \geq \epsilon n$ such that $\sigma(Y) \geq \frac{\epsilon^3 n}{2}$.*

The lemma is just making the link between lemma (3.5) and definition (3.4). In, other words it is saying that if condition (2a) is satisfied, then the partition cannot be declared ϵ-regular.

Lemma 3.7 [1] - *Fix k and γ and let $G = (V, E)$ be a graph with n vertices. Let P be an equitable partition of V into classes $C_0, C_1, ..., C_k$.. Assume $|C_1| > 4^{2k}$ and $4^k > 600\gamma^{-5}$. Given proofs that more than γk^2 pairs (C_r, C_s) are not γ-regular, then one can find in $O(n)$ time a partition P' (which is a refinement of P) into $1 + k4^k$ classes, with the exceptional class of cardinality at most $|C_0| + \frac{n}{4^k}$ and such that*

$$ind(P') \geq ind(P) + \frac{\gamma^5}{20}. \tag{3.9}$$

What the lemma is basically saying is: look for each pairs in the partition. If they are regular then it is okay, otherwise refine the partition into a new partition which has better index (the index of the new partition is at least $\frac{\gamma^5}{20}$ greater than the index of the previous partition. The proofs (or certificates) that some pair is not ϵ-regular are provided by Lemma (3.5).

3.1.2 Alon algorithm

Now, the algorithm is a trivial matter of arranging the above lemmas and the constructive version of the regularity lemma [1].

Given any $\epsilon > 0$ and a positive integer t, we define the constants $N = N(\epsilon, t), T = T(\epsilon, t)$ as follows; let b be the least positive integer such that

$$4^b > 600(\frac{\epsilon^4}{16})^{-5}, b \geq t. \tag{3.10}$$

Let f be the integer valued function defined inductively as

$$f(0) = b, f(i+1) = f(i)4^{f(i)}. \tag{3.11}$$

Put $T = f(\lceil 10(\frac{\epsilon^4}{16})^{-5} \rceil)$ and put $N = max\{T4^{2T}, \frac{32T}{\epsilon^5}\}$. The algorithm for a graph $G = (V, E)$ (with n vertices), is as following:

1) Randomly divide the vertices of G into an equitable partition P_1 with classes $C_0, C_1, ..., C_b$ where $|C_1| = \lfloor \frac{n}{b} \rfloor$ and hence $|C_0| < b$. Denote $k_1 = b$.

2) For every pair (C_r, C_s) of P_i, verify if it is ϵ-regular or find $X \in C_r$, $Y \in C_s$, $|X| \geq \frac{\epsilon^4}{16} |C_1|$, $|Y| \geq \frac{\epsilon^4}{16} |C_1|$, such that $|d(X, Y) - d(C_s, C_t| \geq \epsilon^4$.

3) If there are at most $\epsilon \binom{k_i}{2}$ pairs that are not verified as ϵ-regular, then halt. P_i is an ϵ-regular partition.

4) Apply the refinement algorithm (Lemma 3.7) where $P = P_i$, $k = k_i$, $\gamma = \frac{\epsilon^4}{16}$ and obtain a partition P' with $1 + k_i 4^{k_i}$ classes.

5) Let $k_{i+1} = k_i 4 k_i$, $P_{i+1} = P'$, $i = i + 1$, and go to step (2).

What the algorithm is essentially saying is: divide the vertices of G into an arbitrary number of classes, check if each pair is regular or not (if not, save the certificates and complements), then if the number of non-regular pairs is greater than some threshold (depended on ϵ), refine the algorithm (for each pair separate the certificate from the complement). Repeat this procedure, until the partition becomes regular. With each iteration the index is guaranteed to increase for at least some constant $\frac{\epsilon^4}{16}$.

3.2 Frieze-Kannan Algorithm

Alon algorithm isn't the only constructive version of the Regularity Lemma. In 1999, Alan Frieze and Ravi Kannan, gave an another version of the lemma which is based on the singular value decomposition [18]. In essence, the difference between algorithms is on step (2) when the algorithms are completely different.

3.2.1 Building the algorithm

A m x n matrix $\mathbf{A}$ has a *Singular Value Decomposition* into the sum of rank one matrices [21].

Definition 3.8 - *The first singular value σ_1 is defined as*

$$\sigma_1(\mathbf{A}) = max_{|x|=|y|=1} \left| x^T \mathbf{A} y \right| \tag{3.12}$$

This value can be computed in polynomial time and it is the square root of the largest eigenvalue of $\boldsymbol{A}^T \boldsymbol{A}$.

Definition 3.9 - *Let $\boldsymbol{A}$ be the adjacency matrix of G. Suppose we now have a partition of V into V_1, $V_2, ...$ and we wish to check whether (V_r, V_s) form a γ-regular pair for some γ. Set $R = V_r$, $C = V_s$ and let $A_{r,s}$ be the R x C submatrix of A corresponding to these rows and columns. Let*

$$d = \frac{1}{|V_r|\,|V_s|} \sum_{i \in V_r} \sum_{j \in V_s} \mathbf{A}(i,j) \tag{3.13}$$

be the average of the entries in $A_{r,s}$. Let $\boldsymbol{D}$ be the R x C matrix with all the entries equal to d. Let $\boldsymbol{W} = \boldsymbol{W}_{r,s} = \boldsymbol{A}_{r,s} - \boldsymbol{D}$. (V_r, V_s) is an ϵ-regular pair iff

$$|\mathbf{W}(S,T| \leq \epsilon\,|S|\,|T| \tag{3.14}$$

for all $S \subseteq R$, $|S| \geq \epsilon\,|R|$, $T \subseteq C$, $|T| \geq \epsilon\,|C|$.

The following lemma links the previous two definitions to each other.

***Lemma 3.10** [18] - Let $\boldsymbol{W}$ be an R x C matrix with $|R| = p$, $|C| = q$ and $||\,\boldsymbol{W}||_\infty \leq 1$, and γ be a positive real.*

(a) If there exist $S \subseteq R$, $T \subseteq C$ such that $|S| \geq \gamma p$, $|T| \geq \gamma p$ and $|\,\boldsymbol{W}(S,T)| \geq \gamma\,|S|\,|T|$ then $\sigma_1(\boldsymbol{W}) \geq \gamma^3\sqrt{pq}$.

(b) If $\sigma_1(\boldsymbol{W}) \geq \gamma\sqrt{pq}$ then there exist $S \subseteq R$, $T \subseteq C$ such that $|S| \geq \gamma' p$, $|T| \geq \gamma' q$ and $|\,\boldsymbol{W}(S,T) \geq \gamma'\,|S|\,|T|$ where $\gamma' = \frac{\gamma^3}{108}$. Furthermore S, T can be constructed in polynomial time

3.2.2 Frieze-Kannan Algorithm

Having given the relation between the singular values and the regularity pairs, the algorithm now is straightforward.

1) Randomly divide the vertices of G into an equitable partition P_1 with classes $C_0, C_1, ..., C_b$ where $|C_1| = \lfloor \frac{n}{b} \rfloor$ and hence $|C_0| < b$. Denote $k_1 = b$.

2) For every pair (V_r, V_s) of P_1, compute $\sigma_1(\mathbf{W}_{r,s})$. If the pair (V_r, V_s) are not ϵ-regular then by Lemma (3.10) we obtain a proof that they are not $\gamma = \frac{\epsilon^8}{108}$-regular.

3) If there are at most $\epsilon\binom{k_i}{2}$ pairs that are not verified as ϵ-regular, then halt. P_i is an ϵ-regular partition.

4) Apply Lemma (3.7) where $P = P_i$, $k = k_i$, $\gamma = \frac{\epsilon^9}{108}$ and obtain a partition P' with $1 + k_i 4^{k_i}$.

5) Let $k_{i+1} = k_i 4^{k_i}$, $P_{i+1} = P'$, $i = i + 1$ and go to step (2).

The algorithm finishes in t most $O(\epsilon^{-45})$ steps with an ϵ-regular partition, since $ind \leq 1/2$ and each non-terminating step increases the index by $\gamma^5/20$. For a relatively small ϵ, like $\epsilon = 0.06$ then the number of steps we need in order to get guaranteed a regular partition is 1.53 x 10^{54}.

3.3 Heuristics

We saw in the previous section that the number of steps for which the algorithm guarantees a regular partition, is very large, far larger than something we can compute even in a supercomputer. Furthermore, considering that in each step the graph gets divided into $1 + k_i 4^{k_i}$ classes (when i is upper bounded by the number $1.53 * 10^{54}$ means that the algorithm needs a graph which is astronomically large. While, innocently it looks that the algorithm has a good complexity $O(n^{2.376})$, the suppressed constant in the big-Oh is an exponential tower. While the researchers were looking for ways to address the issue, Gowers proved that the tower is necessary for the algorithm to work [22]. This has deep consequences because it shows that an exact algorithm must have very high complexity in order to work, and for some time made the lemma completely impractical for computational usage.

Perhaps the first time the regularity lemma was used in practical Computer Science was here at Ca' Foscari, in a thesis of Anna Sperotto, which later resulted in a paper by Sperotto and Pelillo [44]. They used the regularity lemma as a pre-processing step to make the process of clustering (dominant set based clustering) better and more efficient. A few years later, as a result of a thesis of Trivedi at Worcester Polytechnic Institute, there was a paper published by Sarkozy, Song, Szemerédi and Trivedi when they used the regularity lemma in the same way to make the process of spectral clustering better and more efficient [40]. Both papers use almost identical heuristics. Here, we will combine the heuristics used in those papers, while completely changing one of the heuristics, and also creating a new heuristic.

Heuristic 1: On the original algorithm, we find all possible ϵ-regular pairs in the graph, and then we divide the pairs into complements and certificates. As consequence, each class may be involved with up to $(k-1)$ ϵ-regular pairs. In our implementation, we will decrease the cardinality of atoms in each iteration by considering at most a single ϵ-irregular pairs that involves the given class. Basically, if a class forms ϵ-regular pairs with every other class, we leave it as it is. Otherwise, we randomly choose a single ϵ-irregular pair that involves that class, and work for it. We note that a random choice likely isn't the best model, but further investigation is needed to get better strategies on choosing the pair.

Heuristic 2: On the original algorithm, in each step the cardinality of a class is divided by 4^k (k is the number of classes in the previous iteration) which means that in order for the algorithm to work, the graph should be extremely large, and so it limits the algorithm. Here, we decide a parameter l (typically 2, 3 or 4) and the cardinality of the new classes is depended on l. Specifically, the cardinality of the new classes will be $\frac{|V_i|}{l}$, while the number of classes on each iteration will be the number of classes in the previous iteration times l. This achieves control in the number of classes, while also it ensures that the cardinality of classes doesn't become too small very fast.

Heuristic 3: While the previous heuristic fixes some problems, on the other side, it might potentially fast increase the size of the exceptional class. Because the cardinality of the exceptional class must be at most the same as the number of classes, we need to deal with this problem. In the two papers we mentioned, the vertices were recycled by putting them in the other classes. While this fixes the problem of a large exceptional class, it might make the other classes even more irregular than in the previous iteration. Specifically, if the vertexes recycled are more ϵ-irregular to the classes they got appointed compared with the vertexes which were removed from those classes, this makes the process worse and in fact, it decreases the index of the entire partition. In this implementation, we used a new approach. Instead of recycling the vertexes randomly to the other classes, we created new classes from those vertexes. While these new classes are highly irregular (and so they make the number of irregular pairs higher), we know that in the next iteration they will be refined. It needs to be said that both approaches have pros and cons, and at the moment we can't say which approach performs better. Furthermore, it looks likely that by intelligently deciding in what classes to put the specific vertexes (if we use the first approach), or what vertexes to join together in the new classes (if we use the second approach), we might get better results. An idea might be computing the density of classes and then on that class recycling the vertexes who have the closest degree to that density (first approach), or putting the vertexes with close degree in the same class (second approach). While at this stage we cannot say it with confidence, it is worth further investigating this.

Heuristic 4: Even with the above modification, it is possible that the algorithm doesn't converge to a regular partition. We use a parameter $h = 2 * l$ and we decide that if the cardinality of classes is smaller than h, we halt. Then we compute the ratio between number of allowed irregular pairs, and the number of actual irregular pairs. The ratio of a regular partition is lower bounded by 0 (only if there is no irregular pair in the partition) and 1 (if the number of irregular pairs is exactly the same as the number of allowed irregular pairs). In the case when we cannot find a regular partition, we look at the result of this ratio. If the ratio is very close to 1, we still count the partition as regular and continue working on the other stage. If not, then we decide that the algorithm has failed. The rationalization behind this is that a hard coding threshold makes sense in theory, but a partition which has 0.99 as ratio (and so is counted as regular) has pretty much the same quality as a partition which has 1.01 as ratio (and so is counted as irregular). Despite that the second partition isn't technically regular, it can be considered as 'quasi-regular' and so might be useful to use it.

Finally, we need to say that even with these heuristics, the algorithm doesn't work for all graphs. However, from empirical evidence we can say that it works for *most* graphs. We will show in Chapter 5 how you can get a regular partition (or almost regular) from different types of graphs.

3.4 Using the Regularity Lemma

We talked so far for the Regularity Lemma and its algorithm. Now, we will explain how we can actually use the lemma.

Definition 3.11 - *Given an ϵ-regular partition of a graph G = (V,E), the reduced graph R is the graph which as vertexes have the classes of G, while as edges has ϵ-regular partitions on G with density above some threshold d. The reduced graph R is considered as the essence of graph G, and it inherits its main properties.*

A picture of the previous definition, as given in [44] can be seen:

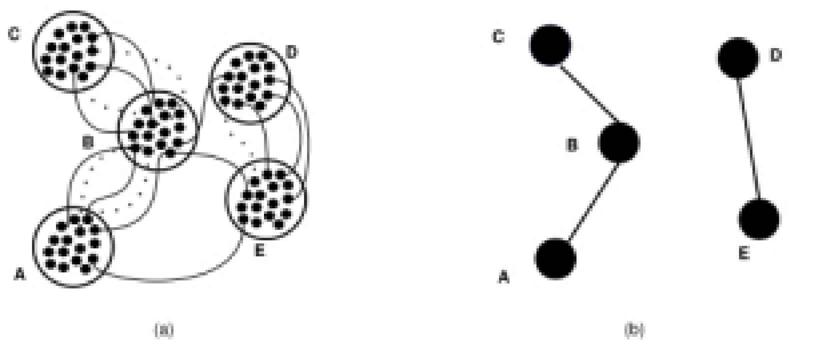


FIGURE 3.1: Example of using the regularity lemma. (a) The original graph; (b) The reduced graph

So, by just simply considering the classes of the original graph as vertexes in the reduced graph, and regular pairs with considerable density of the original graph as edges in the reduced graph, we obtain a small graph (compared to the original graph) which inherits the main properties of the original graph (for example every clique in the R is also contained in G).

Because the adjacency matrix of the reduced graph is very sparse, we have decided to relax the constraints, and to consider an edge any pair whose density is above some threshold d (we don't require the pair to be regular anymore). The rationalization behind it is as following: by using the lemma, similar vertexes are already grouped together in classes. That means that we already have a good knowledge of the regularity structure on the graph, so we don't need to see if the pairs between classes are regular or not. Again, this is almost a heuristic measure (it deviates from the definition) so further investigation might be needed. However, [40] showed empirical evidence that the results are better this way, by forcing some edges between classes.

While the regularity lemma shows how to reduce the graph, it doesn't show how to expand it. This can be done by so-called Key Lemma [28], [29] and [27].

Theorem 3.12 (Key Lemma) - *Given the reduced graph R, $d > \epsilon > 0$, a positive integer m, construct graph G^* which is an approximation of graph G by performing these two steps:*

1. *replace every vertex of R by m vertices.*
2. *replace the edges of R with regular pairs of density at least d.*

The lemma is just the inverse of definition (3.11). It shows how we can expand the reduced graph R into an approximation of the original graph G. The procedure is quite simple: for every class on R create m vertexes on G (m is the cardinality of classes) and put edges between those vertexes only if classes form regular pairs with density above some threshold d. Of course, if we use the relaxed version of the definition (3.11) when we don't look for regular pairs, we should relax the key lemma too and put edges between classes which form pairs with density d, regardless if the pair is regular or not.

Now that we gave the key lemma, we have all the tools to provide how we can use those theorems:

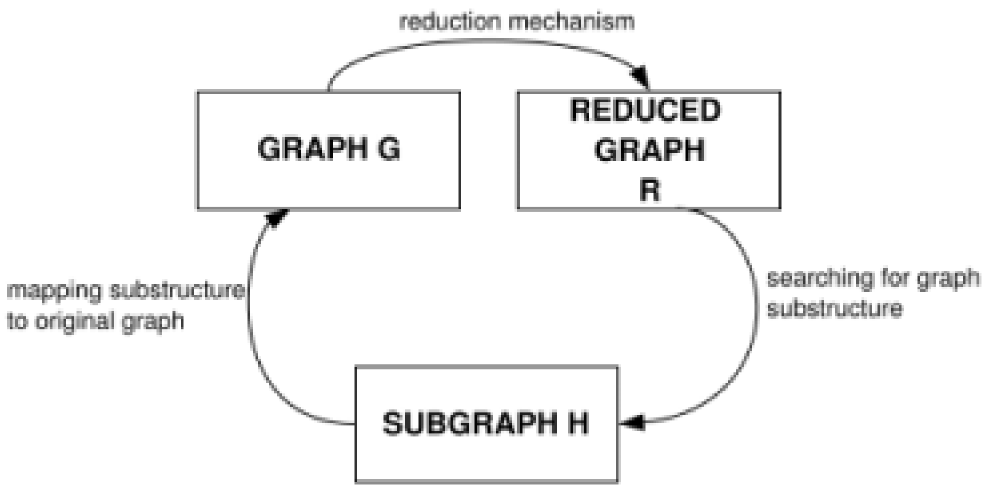
1) For a graph $G = (V, E)$ perform the regularity lemma and so find a regular partition.

2) Using the definition (3.11) construct the reduced graph R.

3) Study the reduced graph R, knowing that it inherits the properties of the original graph G (specifically in this thesis, by study we mean perform graph and subgraph isomorphism, in general it means to analyze the properties of R and its subgraphs).

4) Expand the graph R into graph G^* which is an approximation of graph G, knowing that it has the same properties of R (every small subgraph of R is also a subgraph of G).

We conclude this section by showing a picture which shows the above procedure [44]:



reduction mechanism
GRAPH G
REDUCED GRAPH R
searching for graph substructure
SUBGRAPH H
mapping substructure to original graph

FIGURE 3.2: Reduction strategy to find substructures in a graph

3.5 Weighted Version

Czygrinow and Rodl [13] introduced an algorithmic version of the lemma for the weighted graphs. The main change is that the density between a pair of graphs, now needs to deal also with the weights.

Equation (2.2) for the unweighted version of the lemma is replaced with the following equation:

$$d = d_w(A,B) = \frac{\sum_{i=1}^{|A|} \sum_{j=1}^{|B|} w(a_i, b_j)}{|A|\,|B|} \tag{3.15}$$

The equations are very similar. The only difference is that equation (2.2) takes into account only the presence or absence of an edge, while equation (3.15) takes into account also the weight of an edge.

Sperotto and Pelillo [44] extended the idea into taking in account also the intra-class similarities. Remember that the regularity lemma considers only the similarity between the classes (it is formulated in that way, that the graph becomes the union of bipartite graphs, something that doesn't happen with the relaxed version). To consider the similarity between the vertexes of the same class, we need a new rule (this could be easily called **Heuristic 5**).

What we do, is consider the average weighted degree:

$$deg_S(i) = \frac{1}{|S|} \sum_{j \in S} w(i,j), S \subset V. \tag{3.16}$$

Then we sort the vertexes of S in decreasing order. The idea is that when the subset gets repartitioned in the following iteration, the vertexes which have around the same intra-class similarity, will be grouped together. We choose the decreasing order, because this ensures that only the vertexes which are weakly connected join the exceptional class (if we did the other way around, then vertexes which are strongly connected would have joined the exceptional class).

Chapter 4

Graph Isomorphism

The graph isomorphism is one of those problems for which we don't know yet in what class of computational complexity they belong. Since the inception of the problem, there was a belief that the problem lies between the P and NP-complete classes. However, during the time of writing this thesis (November 2015), Lezlo Babai claimed that he has solved the problem in quasi-polynomial time. There is yet to be published a paper on it, but an interesting lecture can be found at [3].

4.1 The Problem of Graph Isomorphism

Definition 4.1 - *For an undirected graph $G = (V, E)$ the order of it is the numbers of vertices, while the size of it is the number of edges. Two vertices $i, j \in V$ are said to be adjacent if there is an edge between them. The adjacency matrix of the graph G is an n by n symmetric matrix, is defined as usual [9]:*

$$a_{ij} = \left\{ \begin{array}{ll} 1, & \text{if (i,j) in E,} \\ 0, & \text{otherwise.} \end{array} \right\}$$

Definition 4.2 - *Given two graphs $G' = (V', E')$ and $G'' = (V'', E'')$, an isomorphism between them is any bijection $\phi : V' \to V''$ such that $(i, j) \in E' \Leftrightarrow (\phi(i), \phi(j)) \in E''$, for all $i, j \in V'$ [36]. The graph isomorphism problem is the problem of deciding if two graphs are isomorphic, and if they are, to find an isomorphism between them.*

As mentioned in the introduction of this chapter, so far there are no algorithms which can solve the problem in polynomial times, and even analyzing the complexity of the problem hasn't been possible.

Definition 4.3 - *A clique of an undirected graph $G = (V, E)$ is a subset of vertexes $V' \subset V$ where each edge $i \in V'$ is adjacent with any edge $j \in V'$. A clique is said to be maximal, if it isn't contained in any other clique. A clique is said to be maximum if it is the biggest maximal clique in the graph. The clique number is the number of vertexes in the clique*

While we don't know the complexity of graph isomorphism, we know that the maximal clique problem is NP-hard [19]. Furthermore, just finding the clique number has been proven to be intractable for large graphs.

4.1.1 The Association Graph

The idea of the approach used here (and also in [36], [6]) is to transform the problem of graph isomorphism into the problem of finding the maximal clique in a graph. Despite that it might look strange to transform a problem into a significantly more complex and difficult problem, we will show that this works in practice because there are efficient algorithms which can solve the maximal clique problem in reasonable time.

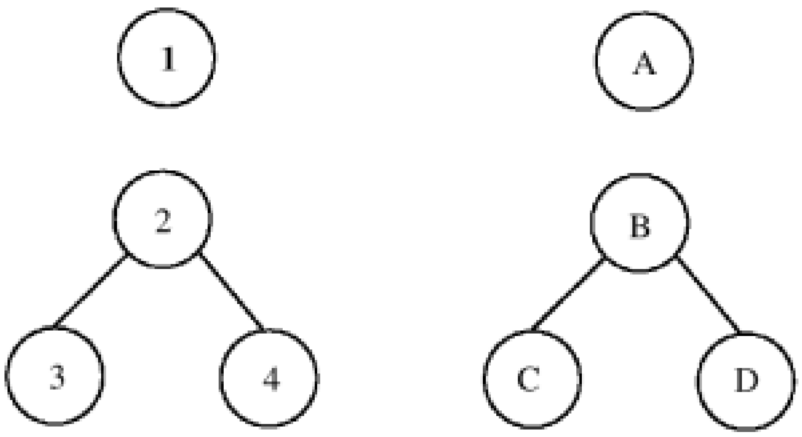


Figure 4.1: A pair of isomorphic graphs

Barrow and Burstall [5] and Kozen [30] introduced the notion of an association graph in order to be able to solve the graph and subgraph isomorphism problem.

Definition 4.4 - *The association graph derived from graphs $G' = (V', E')$ and $G'' = (V'', E'')$ is the undirected graph $G = (V, E)$ where $V = V'xV''$ and*

$E = \{((i,h),(j,k)) \in VxV : i \neq j, h \neq k \text{ and } (i,j) \in E' \Leftrightarrow (h,k) \in E''\}$.

The adjacency matrix $A = (a_{ij,jk})$ of the association graph can be defined as:

$$a_{ih,jk} = \begin{Bmatrix} 1 - (a'_{ij} - a''_{hk})^2, & \text{if } i \neq j \text{ and } h \neq k, \\ 0, & \text{otherwise.} \end{Bmatrix}$$

The adjacency matrix of the association graph created by the two graphs in Figure (4.1) is:

TABLE 4.1: Adjacency Matrix of the association graph created by the graphs in Figure (4.1)

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

Theorem 4.5 [36] - *Let $G' = (V', E')$ and $G'' = (V'', E'')$ be two graphs of order n, and A be their corresponding association graph. G' and G" are isomorphic only if the clique number $\omega(A) = n$. In this case, any maximum clique of A induces an isomorphism between the two graphs G' and G", and vice versa. In general there is an one-to-one correspondence between maximal/maximum cliques in A with maximal/maximum common subgraph isomorphism between G' and G".*

The theorem just establishes the relation between the maximum clique in the association graph, with the isomorphism of the two graphs. The clique number must be the same as the order of the graphs in order for those graphs to be considered isomorphic. If the clique number is lower, then the graphs aren't isomorphic but they have a common subgraph. For our graphs in Figure (4.1), the clique number must be 4.

As mentioned in the beginning of the section, the maximum clique problem is NP-hard. Precise solutions like the Bron-Kerbosch [7] algorithm have a high complexity $O(3^{n/3})$ and cannot be used in practice for even relatively small graphs. The rest of this chapter is dedicated to finding a heuristic algorithm which can give an approximate solution in efficient time.

4.1.2 Continuous Formulation of the Maximum Clique Problem

For an undirected, unweighted graph $G = (V, E)$ of order n, we denote the standard simplex S$_n as$:

$$S_n = \{x \in \mathbb{R}^n : x_i \geq 0 \ \ for \ \ all \ \ i = 1, ..., n, \ \ and \ \sum_{i=1}^{n} x_i = 1\}.$$

Given a subset of vertices C of G we denote by x^c its characteristic vector, which is a point in S_n defined as

$$a_{ih,jk} = \left\{ \begin{array}{cc} 1/|C|\,, & if \ i \in C \\ 0, & \text{otherwise} \end{array} \right\}$$

Now, consider the following quadratic function,

$$f(x) = x^T Ax = \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij}x_i x_j \qquad (4.1)$$

where $A = (a_{ij})$ is the adjacency matrix of G. A point $x^* \in S_n$ is called a global maximizer of f in S_n if $f(x^*) \geq f(x)$, for all $x \in S_n$, while it is called a local maximizer if there exists an $\epsilon > 0$ such that $f(x^*) \geq f(x)$ for all $x \in S_n$ whose distance from x* is less than ϵ.

The Motzkin-Straus theorem [33] establishes a connection between global (local) maximizers of the function f in S_n and maximum (maximal) cliques of G. In particular, it says that that C is a maximum clique in G, if and only if its characteristic vector x^c is a global maximiser of f in S_n. This result was extended by Pelillo and Jagota [37] and Gibbons [20] which shows a similar relation between strict local maximizers and maximal cliques. The theorem is very important because it allows us to change from district to continuous domain, in effect, transforming the problem from clique finding (NP-hard) to the problem of finding a maximizer for the function f in S_n. As consequence, for a long time it has served as the basis for many clique finding procedures.

However, the theorem comes with a problem. As observed in [34] and later formalized by Pelillo and Jagota in [37], the theorem in the original form allows the existence of spurious solutions. While spurious solutions are global maximizers they represent a problem because they do not allow to extract the clique from them. Saying that, there is a direct solution to this problem presented by Bomze in [6]. The solution is the regularized version of f,

$$f(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j + \frac{1}{2} \sum_{i=1}^{n} x_i^2 \tag{4.2}$$

which can be obtained from equation 4.1 by simply adding 1/2 in the diagonal of the adjacency matrix A.

4.1.3 A quadratic program for graph isomorphism

Having two graphs G' and G" of order n, and building from them the association graph G with adjacency matrix A, the graph isomorphism problem becomes equivalent of solving the following program:

$$maximize f(x) = x^T(A + \frac{1}{2} I_N)x \quad subject\ to\ x \in S_N \tag{4.3}$$

Theorem 4.6 [36] - *Let $G' = (V', E')$ and $G'' = (V'', E'')$ be two graphs of order n. Let x^* be a global solution of program (4.3). G' and G" are isomorphic if and only if*

$f(x^*) = 1 - \frac{1}{2n}$. *In this case, any global solution of the equation (4.3) corresponds to an isomorphism between G' and G" (with vice versa holding true).*

After this formulation, then the Motzkin-Straus function f becomes:

$$f(x) = \sum_{i,h} \sum_{j \neq i} \sum_{k \neq h} a'_{ij} a''_{hk} x_{ih} x_{jk} + \sum_{i,h} \sum_{j \neq i} \sum_{k \neq h} (1 - a'_{ij})(1 - a''_{hk}) x_{ih} x_{jk} + \frac{1}{2} \sum_{i,h} x_{ih}^2. \tag{4.4}$$

4.2 Replicator Equations and Graph Isomorphism

Replicator equations have been developed and studied in the field of evolutionary game theory by Maynard Smith [43], while later were intensively studied by Weibull [50]. In essence, they try to model the evolution of animal behavior using the tools of game theory. The idea is to consider a large population of individuals as players belonging to the same specie and competing for a limited resource, such as food or territory. Compared to classical game theory, here players do not act rationally, but act according to their genetically pre-programmed behavior pattern, or pure strategies.

At time t, the average payoff of strategy i is given by:

$$\pi_i(t) = \sum_{j=1}^{n} w_{ij} x_j(t)$$

where w_{ij} represents the payoff of an individual playing strategy i against an another player who is playing strategy j.

The mean over the entire population is simply:

$$\sum_{i=1}^{n} w_{ij} x_j(t).$$

The assumption is that the game is played repeatedly, and that natural selection will result in the evolution of the best strategy. This can be described by the following differential equations [47]:

$$\frac{dx_i(t)}{dt} = x_i(t)(\pi_i(t) - \sum_{j=1}^{n} x_j(t)\pi_j(t)), \quad i = 1, ..., n \tag{4.5}$$

The idea behind the model is that the average rate of increase $\frac{dx_i(t)}{x_i(t)}$ equals the difference between the average fitness of strategy i and the mean fitness over the entire population.

The discrete version of the previous equation is as follows:

$$x_i(t+1) = \frac{x_i(t)\pi_i(t)}{\sum_{j=1}^{n} x_j(t)\pi_j(t)}, \quad i = 1, ..., n \tag{4.6}$$

The equation (4.6) is the central equation of this chapter, and in essence, the entire idea of the algorithm is performing this equation until it converges into a stable solution. That is guaranteed by the following theorem.

Theorem 4.7 - The fundamental theorem of natural selection [50], [12] - *Given a non-negative symmetric payoff matrix W, the quadratic polynomial F defined as*

$$F(x) = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} x_i x_j \tag{4.7}$$

is strictly increasing along any consistent trajectory of both continuous-time (equation 4.5) and discrete-time (equation 4.6) replicator dynamics. Its derivative with respect to time is positive:

$$\frac{d}{dt} F(x(t)) > 0$$

for the continuous case, and

$$F(x(t+1)) > F(x(t))$$

for the discrete case, unless x(t) is a stationary point. Furthermore, any such trajectory converges to a unique stationary point.

4.3 The Algorithm

On this sub-chapter, we will put all the results we gained in the previous parts of this chapter together, and in doing so, we will gain an efficient algorithm which finds an isomorphism between graphs in polynomial time.

Having two graphs, $G' = (V', E')$ and $G' = (V'', E'')$ with order n, we denote the adjacency matrix A of the association graph and by letting:

$$W = A + \frac{1}{2} I_N$$

where I is the identity matrix, we know that the replicator dynamics, starting from an initial vector, will maximize the function $f(x) = X^T(A + \frac{1}{2}I_N)x$ in the standard simplex S_N and will eventually converge to a strict maximizer. Theorem (4.5) tells that the maximum clique extracted from the characteristic vector is in one-to-one relation with an isomorphism between the two graphs G' and G". While, in theory it isn't guaranteed that the dynamics will ever converge, the experimental results in [36], [6] has shown that there are many global maximizers, and that the algorithm frequently converges to one of them.

All that remains to do, is to decide how to initialize the characteristic vector, and a plausible solution is the vector $(\frac{1}{N}, \ldots \frac{1}{N})^T$, which represents the barycenter of the simplex. While this ensures that no solution if favored, on the other side it creates a symmetry problems and the algorithm might get stuck in the beginning. A straightforward solution is to add a little perturbation in order to break the symmetry.

Now, we can put the pieces together and give the following algorithm (in order to get an isomorphism between graphs).

For two undirected, unweighted graphs, $G' = (V', E')$ and $G'' = (V'', E'')$ an isomorphism between them can be gained by performing the following actions:

1) Compute the adjacency matrix of their association graph as shows in section (4.1.1). Add $\frac{1}{2}$ in the diagonal.

2) Initialize the characteristic vector, near the barycenter of the simplex.

3) Perform iteratively equation (4.6), which should converge to a global maximizer.

4) A maximal clique can be extracted by the final characteristic vector (gained from step 3).

5) Check if the extracted clique is really a clique (not a saddle point). If it is a saddle point, perturb a bit, and go back to step (3)

6) There is an one-to-one correspondence between the maximum clique and an isomorphism between graphs G' and G". If the clique number is the same as the order of the graphs, we have a graph isomorphism, otherwise we have a sub-graph isomorphism.

We conclude this section by showing how the replicator dynamics evolve (Figure 4.2), using the graphs in Figure 4.1 (on the x-axis we have the number of iterations, while on y-axis, we have the components of the state vector). We see that only 50 iterations were needed, and so the entire algorithm in a modern computer takes around a second.

4.4 Faster Replicator Dynamics

The idea of replicator dynamics was extended in a publication by Hofbauer and Weibull [24], and they built a new version of replicator dynamics which work faster than the previous one. The continuous version is given in equation (4.8)

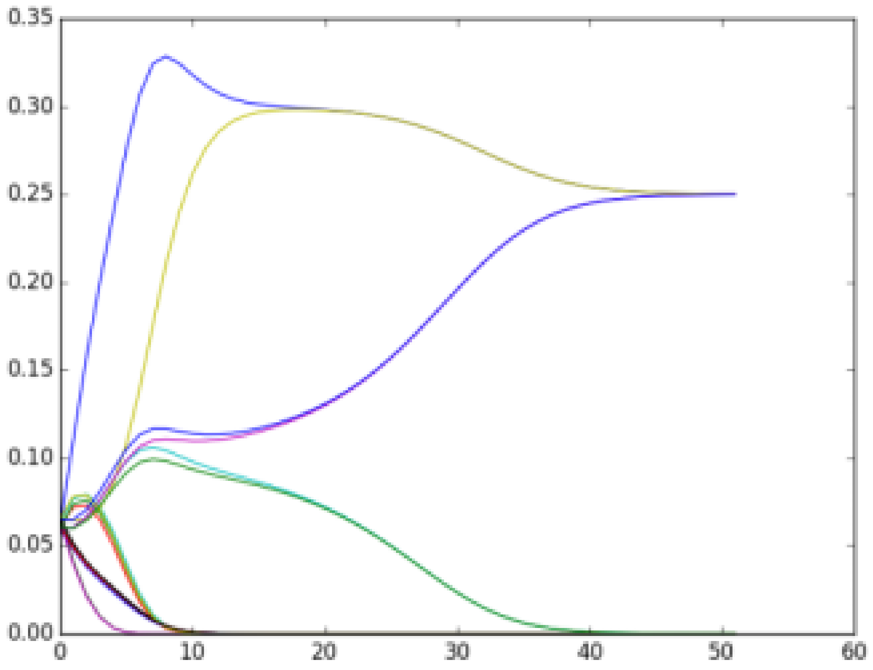


FIGURE 4.2: The evolution of replicator dynamics

$$x_i(t+1) = \frac{x_i(t)e^{K\pi_i(t)}}{\sum_{j=1}^{n} x_j(t)e^{K\pi_j(t)}}, \quad i = 1, ..., n \tag{4.8}$$

The dynamics are payoff monotonic:

$$\frac{x_i(t+1) - x_i(t)}{x_i(t)} > \frac{x_j(t+1) - x_j(t)}{x_j(t)} \Leftrightarrow \pi_i(t) > \pi_j(t). \tag{4.9}$$

K is just a number, which change the way how replicator dynamics behave. For the graph given in Figure 4.1, we see how the dynamics differ while we change the parameter K.

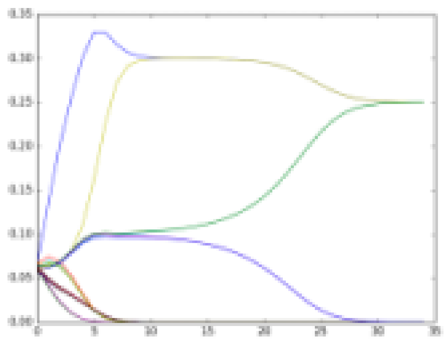


Figure 4.3: The evolution of replicator dynamics when K = 3

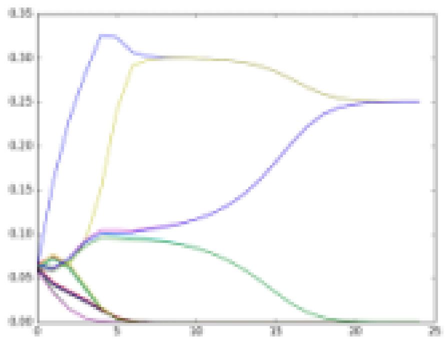


Figure 4.4: The evolution of replicator dynamics when K = 4

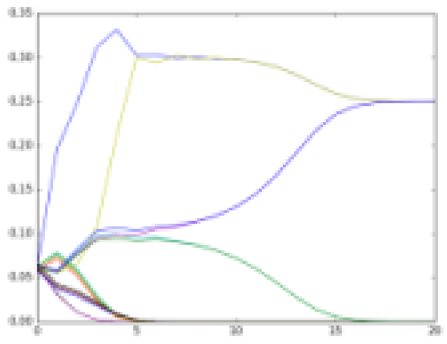


Figure 4.5: The evolution of replicator dynamics when K = 5

We see how differing parameter K differs the number of iterations. While generally for $K < 3$ and for a big K, the exponential replicator dynamics perform worse than the normal replicator dynamics, for the values in between, they seem to completely outperform normal replicator dynamics. The caveat is that the programmer needs to play a bit with the parameter in order to find which K gives the best results.

The algorithm is completely the same as the algorithm in the previous section. For the purpose of this implementation, we are using the new replicator dynamics because of their efficiency.

4.5 Weighted Version

The weighted version of the graph isomorphism is very similar to the unweighted version. The basic idea is to transform the problem of isomorphism into the problem of finding a dominant set in the weighted association graph.

Definition 4.8 - *A non-empty subset of vertices $S \subseteq V$ such that $W(T) > 0$ for any non-empty $T \subseteq S$, is said to be a dominant set if:*

1. $w_S(i) > 0$, for all $i \in S$ (internal homogeneity).

2. $w_{S \cup i}(i) < 0$, for all $i \notin S$ (external homogeneity) [35].

The following theorem generalizes the Motzkin-Strauss theorem from graph theory:

Theorem 4.9 [48]**-** *Evolutionary stable strategies of the clustering game with affinity matrix A are in an one-to-one correspondence with the dominant sets.*

In some way, we can consider dominant sets as generalization of the maximal clique for weighted graphs. We will use replicator dynamics to find dominant sets, with the biggest change being that we won't need to add $\frac{1}{2}$ in the diagonal of the association graph. The equation (4.4) becomes:

$$f(x) = \sum_{i,h} \sum_{j \neq i} \sum_{k \neq h} a'_{ij} a''_{hk} x_{ih} x_{jk} + \sum_{i,h} \sum_{j \neq i} \sum_{k \neq h} (1 - a'_{ij})(1 - a''_{hk}) x_{ih} x_{jk}. \tag{4.10}$$

The other parts of the algorithm stay roughly the same as before.

Chapter 5

Experimental Results

In this chapter, we are going to put the algorithms together and to make detailed experiments on different sets of databases. We will use three types of datasets. The first type of datasets are taken from 'UCI Machine Learning Repository', the second type of datasets are randomly generated, while the third type of datasets are taken from 'Stanford Large Network Dataset Collection' website, which contains graphs of social networks. Those graphs, are by far the largest graphs we are going to use in this thesis. An interesting property of them, are that the graphs are incredibly sparse, and so we will be able to see how the algorithm works in sparse graphs.

First, we will start by performing the regularity lemma of Szemerédi in isolation and see how different responses vary while we change parameter ϵ. Then we will perform the matching algorithm in isolation in order to get convinced that it works, and finally we will put together both algorithms and see the results of matching after we have performed the regularity lemma.

On all three types of datasets, the approach is the same. We get the first graph from the dataset, and then we will do a number of random permutations in the adjacency matrix of the graph in order to get the second graph. In this way, we are sure that the graphs are isomorphic. After we do that, the final step is to find an isomorphism/sub-isomorphism on the graphs, and count the number of matched vertices.

5.1 Testing the Regularity Lemma

On this section we are going to test the regularity lemma on isolation, looking how index progresses on each iteration, and how the number of classes and size of class changes while we change the parameter ϵ. We will also measure the time and the number of iterations needed for the algorithm.

5.1.1 Case Study 1: Ionosphere Dataset

UCI Machine Learning Repository [2] is a collection of databases that are used by the machine learning community and students. Most of the datasets are feature-based and donated from various researchers.

Because the datasets are feature-based, we need a way of turning them into graphs. Our approach is using some similarity measure between vectors, and then based on some parameter δ, thresholding them into ones and zeros. For similarity measure we used the Euclidean distance $w(i,j) = exp(\frac{-||v_i - v_j||^2}{\sigma^2)})$ between corresponding attribute vectors.

For this section, we're going to use the Ionosphere Dataset [42], which is a feature-based dataset containing 351 instances, each one of them having 34 continuous attributes in addition to response attribute, which is a discrete attribute being either 'good' or 'bad'. There are no missing attributes in the dataset. It is one of the most commonly used datasets in machine learning, as can be seen by the number of papers which have cited it.

On the following tables we are going to give the final index, number of classes, size of class, size of exceptional class, number of iterations, time needed to perform the algorithm, number of times we managed to find a regular partition (a number between 0 and 1, 0 means never while 1 means always) and how close we were on finding a regular partition which is the ratio between number of irregular pairs and the maximum number of allowed irregular pairs [1].

[1]The regularity lemma allows some irregular pairs based on parameter ϵ. If the ratio between number of irregular pairs and the maximum number of allowed irregular pairs is less or equal to 1, it means that we have got a regular pairs, while if the ratio is greater than 1, then it means that the partition is irregular. We measure this because while theoretically, if we don't manage to get a regular partition it means that the experiment has failed, in reality it is is possible to get some information from them. A concrete example would be finding a partition with ratio 1, and an another partition with ratio 1.01. While the second partition is irregular, the difference in quality between it and the first partition (which is regular) is completely negligible

Table 5.1: Ionosphere: Mean value of response variables, affected by the change of parameter ϵ.

mean / ϵ	0.2	0.3	0.4	0.5	0.6
Final Index	0.2	0.2	0.19	0.17	0.13
Number of classes	175	175	112	31.2	8
Size of class	2	2	3.8	12.2	43
Size of Exceptional Class	1	1	1	3.8	7
Number of iterations	7	7	6.4	4.8	3
CPU time (sec)	4.42	4.18	4.76	1.12	0.4
Successful	1	1	1	1	1
Ratio	0.9	0.6	0.76	0.88	0.76

We did ten Monte Carlo simulations (each based on random initial partitioning) for every ϵ, in order to get the results (mean and standard deviation for each of the variables mentioned). We see that the final index is inversely proportional with ϵ (index decreases as epsilon increases). This completely makes sense, because we know that the smaller ϵ is, the better partition is. Directly, this leads us to the number of iterations, and we see that for small ϵ, we need more iterations than for a big ϵ (the same goes for the time). The number of classes for small ϵ is 175, which means that we have got 2:1 compression rate. While we increase the ϵ, we see that the number of classes decreases, and for a large $\epsilon = 0.6$, the number of classes is only 8 (compression rate is 44:1). On the other hand, the size of class goes the other way around, small ϵ gives small size of classes, while large ϵ gives a large number of classes. In all cases, the size of exceptional class is very small (because of heuristic 3, mentioned in the third chapter). Finally, we see that in all experiments we have succeeded each time on gaining a regular partition, with the ratio between number of irregular pairs and the allowed number of irregular pairs typically being larger for small ϵ than for large ϵ (this can be seen as when we have a small ϵ, there are many irregular pairs, while when we have a large ϵ, there are just a few irregular pairs).

On the table below, we explored the same variables, but now measuring standard deviation. We find that the results are quite robust for small and large ϵ, but when $\epsilon = 0.4$ and $\epsilon = 0.5$ the results between simulations may vary a lot. In particular, for $\epsilon = 0.4$, we see that the standard deviation of number of classes might be extremely large. We justify it by saying that for small ϵ the algorithm needs to purify the graph until the end, while for a large ϵ, it usually takes only a few iterations, and so the graph isn't

Table 5.2: Ionosphere: Standard deviation of response variables, affected by the change of parameter ϵ.

standard deviation /ϵ	0.2	0.3	0.4	0.5	0.6
Final Index	0	0	0	0	0
Number of classes	0	0	51.43	7.6	0
Size of class	0	0	1.47	4.4	0
Size of Exceptional Class	0	0	0	5.6	0
Number of iterations	0	0	0.5	0.4	0
CPU time (sec)	0.18	0.2	2.64	0.26	0.06
Successful	0	0	0	0	0
Ratio	0.03	0.02	0.25	0.06	0.14

partitioned well. But, when the ϵ isn't too small or too large, then based on the initial partitioning, the graph might end with different number of classes.

The final variable we're going to measure, is how the index of the partition varies between each iteration. We expect the initial index to be small, and then with each iteration to increase until it reaches the final value.

Table 5.3: Ionosphere: The increase of index depending on ϵ.

ϵ / ind(P)	ind(P_1)	ind(P_2)	ind(P_3)	ind(P_4)	ind(P_5)	ind(P_6)	ind(P_7)
0.2	0.05	0.08	0.12	0.16	0.17	0.18	0.2
0.3	0.05	0.08	0.12	0.16	0.17	0.18	0.2
0.4	0.05	0.08	0.12	0.16	0.17	0.18	-
0.5	0.05	0.08	0.12	0.16	0.17	-	-
0.6	0.05	0.08	0.12	-	-	-	-

Clearly, the index on each iteration is completely depended on the ϵ parameter. '-' means that the partition has finished by finding a regular partition, so the algorithm doesn't continue in the other iterations.

An interesting thing that we found is that the biggest increase happens in the first four iterations. After those iterations, the index still continues to increase but on a much lower scale. Concretely, we found that the index in the first iteration is just 25% of the final index, after the second iteration it is 40% of the final index, after the third iteration it is 60 % of the final index, and after the fourth iteration it is 80% of the final index. This might be important in cases when compression rate is more important than the quality of the partition, and in those cases we can decide to stop the partitioning algorithm after a few iterations regardless if the graph is regular or not. Clearly, the largest partitioning happens in the first few iterations.

5.1.2 Case Study 2: Randomly Generated Datasets

On this section, we are going to repeat the previous experiments, but this time with 'randomly' generated datasets.

Using completely randomly generated graphs obviously will not work, because the regularity lemma finds the hidden structure in a graph, and so if graphs are random, then there is no such structure (unless the graphs are extremely large, in that case by random chance, there will be some meaningful structures there). Instead, we will use a variation of Condon and Karp graphs [8]. Initially, we will start with a small graph (only a few vertices) and we will decide on the probability between vertices. Then, we will expand the graph into a much larger graph, when each vertex of the original graph will be replaced by a number of vertices, and so, those vertices will now be considered a class. We will connect the classes with edges based on the given probability. Given the graph below:

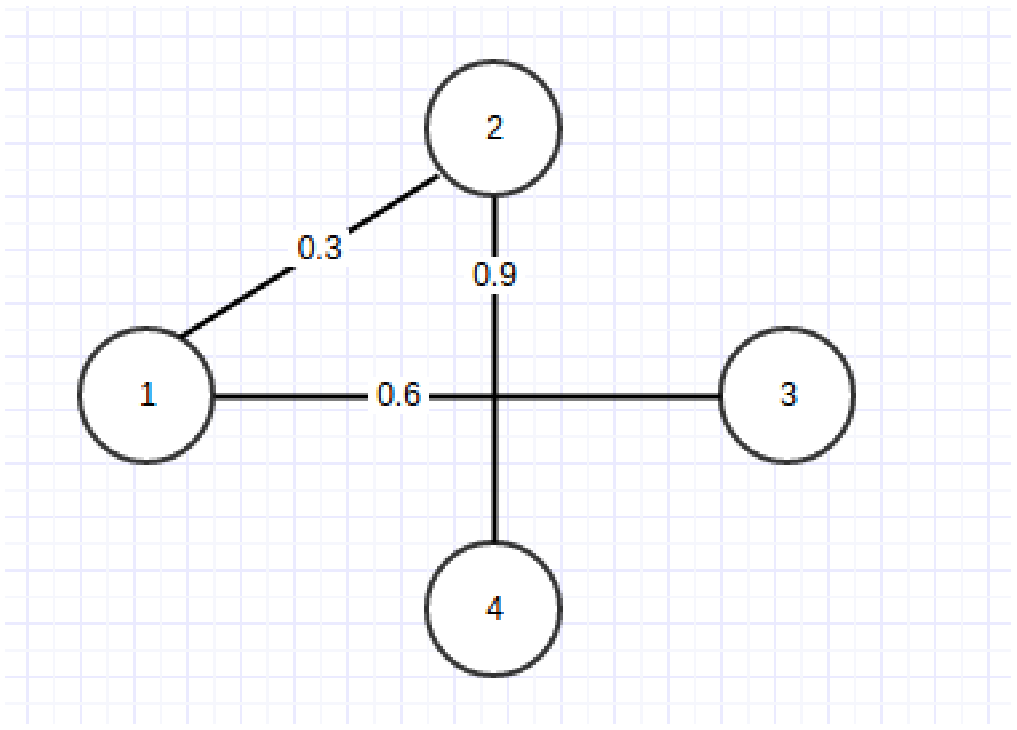


FIGURE 5.1: A small graph

we want to expand it in that way that each vertex will become a class of 200 vertexes. Classes will be connected with edges based on the densities given on the graph. In addition, we are going to add some noise, by allowing some edges (5% chance of each vertex in a class getting connected with a vertex in some class) between classes which shouldn't be connected (on other words, classes 2 and 3, 3 and 4, 1 and 4 will have density 0.05). The graph is relatively sparse, with the number of edges being around $0.12 * n^2$ (n being the number of vertices).

The tables for mean and standard deviation (same approach as in Case Study 1) look:

Table 5.4: Randomly Generated Dataset: Mean value of response variables, affected by the change of parameter ϵ.

mean / ϵ	0.2	0.3	0.4	0.5	0.6
Final Index	0.08	0.08	0.08	0.06	0.02
Number of classes	266	266	266	28.2	2.6
Size of class	3	3	3	61.2	370
Size of Exceptional Class	2	2	2	0.8	0
Number of iterations	8	8	8	4.4	1.4
CPU time (sec)	22.03	27.27	35.19	8.2	1.98
Successful	0	0	1	1	1
Ratio	1.73	1.18	0.89	0.94	0.05

Table 5.5: Randomly Generated Dataset: Standard deviation of response variables, affected by the change of parameter ϵ.

standard deviation /ϵ	0.2	0.3	0.4	0.5	0.6
Final Index	0	0	0	0	0
Number of classes	0	0	0	16.83	1.8
Size of class	0	0	0	69.96	90
Size of Exceptional Class	0	0	0	2.4	0
Number of iterations	0	0	0	1.28	0.6
CPU time (sec)	1.81	4.14	0.09	2.81	1.19
Successful	0	0	0	0	0
Ratio	0.02	0.02	0.01	0.04	0.16

Clearly, the algorithm does a worse job than in case study 1, and is quite affected from the randomness of the data. We see that it for $\epsilon = 0.2$ the number of irregular pairs is significantly than the number of allowed irregular pairs, and so the partition is unacceptable. For $\epsilon = 0.3$ things are a bit better, but still we cannot say that we have a regular partition. For $\epsilon = 0.4$ and for $\epsilon = 054$ we have a regular partition, as we have for bigger ϵ. However, we should be careful and see that for $\epsilon = 0.6$, the algorithm practically does nothing but just separate the graph into two random parts, and because ϵ is large, it declares the graph regular.

So, the only useful ϵ in our example are 0.4 and 0.5, and indeed we can use only ϵ which are around those numbers. We see that for $\epsilon = 0.4$ we have 266 classes, and so a compression ratio of 3:1. Each class has 3 elements in average, while the size of exceptional class is completely negligible. The entire algorithms runs in around half a minute. On the other side, for $\epsilon = 0.5$ we have just 28.2 classes in average, and so a compression ratio of 28.36:1, each class having 70 elements, with the exceptional class being negligible. The algorithm runs for 8 seconds. If we want a better partition, we should choose $\epsilon = 0.4$, however if the compression ratio is the most important thing, then we should choose $\epsilon = 0.5$

We see that the results are quite robust for all epsilons bar $\epsilon = 0.5$.

5.1.3 Case Study 3: Social Networks

With social networks becoming a big part of people's lives, the social network graphs have generated a lot of interest in the community of researchers. The reasons for that is because social networks graphs come from real data, are significantly larger than most of the other types of graphs, and are quite sparse which makes their study quite interesting.

For this evaluation, we are going to use Arxiv HEP-TH (High Energy Physics - Theory) which is a collaboration network from the e-print arXiv which covers the scientific collaboration between authors who have submitted papers to High Energy Physics, in the Theory category [31], taken from the SNAP Dataset Collections [32].

The graph has been collected and studied by Jure Leskovec in the paper cited above. The graph has 9877 nodes and 25998 edges (a node is connected with around 3 other nodes), and so it is a very sparse graph (its density is just $1.46 * 10^{-5}$). Graph is given as an adjacency list. Some of the other characteristics of the graph are:

- the number of nodes in the largest strongly connected components is 8638, while the number of edges in the largest strongly connected component is 24827 (the 1:3 ratio of nodes/edges is inherited here too).

- the average clustering coefficient is 0.47.

- the total number of triangles is 28339.

- the longest shortest path (called also the diameter of the graph) is 17.

Our experiments on this graph are similar to the experiments in the previous two graphs. The main difference is that because of the graph's sparsity, we will need to use significantly smaller ϵ.

Table 5.6: SNAP Dataset: Mean value of response variables, affected by the change of parameter ϵ.

mean / ϵ	0.03	0.05	0.07	0.09
Final Index	2.15e-05	7.5e-06	5.2e-06	6.8e-07
Number of classes	284.8	128	76.8	14.4
Size of class	42	77	138.6	3981.2
Size of Exceptional Class	28.4	21	21	5
Number of iterations	8	7	6.2	2
CPU time (sec)	534.87	397.49	446.45	160.47
Successful	1	1	1	1
Ratio	0.68	0.83	0.8	0.13

Table 5.7: SNAP Dataset: Standard deviation of response variables, affected by the change of parameter ϵ.

standard deviation / ϵ	0.03	0.05	0.07	0.09
Final Index	0	0	0	0
Number of classes	127.62	0	25.6	24.8
Size of class	18.98	0	30.8	1913.6
Size of Exceptional Class	8.24	0	0	8
Number of iterations	0.63	0	0.4	2
CPU time (sec)	133.09	1.4	119.95	236.08
Successful	0	0	0	0
Ratio	0.21	0.1	0.11	0.27

In all cases, the index is extremely small. That is to be expected considering that index is related with the density of the graph. We see that the algorithm has done a good job for every epsilon, finding always a regular partition, and actually being quite deep into the safe zone (the ratio between the number of actual irregular partitions and the maximum number of allowed irregular partition isn't ever greater than 0.83). Depending on the choice of epsilon, we can see how the number of classes (and so the size of class) vary, with the average maximum number of classes being 284.8 for the smallest $\epsilon = 0.03$, while the average minimum of the number of classes being 14.4 for the choice of $\epsilon = 0.09$. The size of class varies from 42 (when $\epsilon = 0.03$, up to 3981, when $\epsilon = 0.09$. The size of exceptional class is typically bigger than in the previous experiments, but still negligible compared with the size of the graph (0.28 % of the size of the graph). The number of iterations seems to vary from 2 to 8, while the time needed to perform the algorithm (tested in a laptop with core-i7 processor) varies from 2.5 up to 9 minutes.

One thing to note is that the standard deviation of the results is quite higher than in the previous experiments, and in fact the results aren't as robust as we would have liked (bar in the case when $\epsilon = 0.05$, which is the only case where the standard deviation is very small for each of the response variables).

5.2 Testing the Isomorphism Algorithm

On this section, we are going to test in details the isomorphism algorithm - using replicator dynamics - explained in Chapter 4. One problem on using this technique is the creation of the association graph. If we want to match two graphs, each of them having m vertices, then the association graph will have m^2 vertices. The adjacency matrix of it will have m^4 entries. For graphs with moderate size, m = 300, creating the association graph will be challenging for a personal computer, because of the memory constraints. Of course, a possible way of dealing with this problem is building the association part by part, but while that removes the problem of space constraints, on the other side it makes the algorithm run slower.

On this section, we are going to start by matching random graphs, and then continue by matching real graphs.

For the experiments on random graphs, we are going to generate the first graph and then do a number of random permutation on its adjacency matrix, in order to get the second graph.

5.2.1 Case Study 1: Matching Random Graphs While the Density Changes

On this section, we are going to match random graphs. In order to perform the experiments faster, we are going to test on graphs with small size (number of vertices = 50). We are going to use different densities on the graph (0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99). For each density we're going to do 10 experiments, and then give the results (as always, we're going to give the mean and standard deviation).

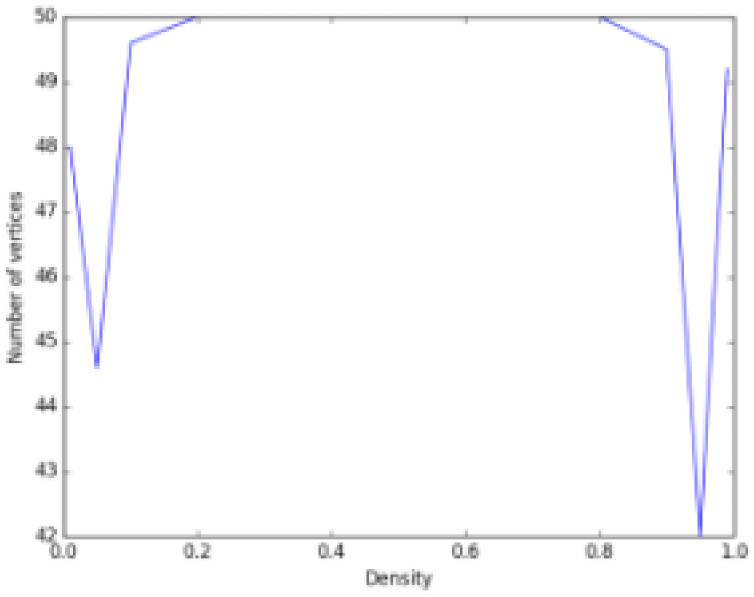


FIGURE 5.2: Mean of the number of matches in a random graph

We see quite clearly, that we get perfect graph isomorphism when the density of the graph isn't on extreme range. So, if the graphs aren't very sparse, or near completely connected, then we get the maximum of matches. However, for graphs that have density less than 10% or more than 90% the results aren't that good, and at times can be very low. This observation completely agrees with [36], where the experiment found similar results (although there were used graphs with 100 vertices).

5.2.2 Case Study 2: Matching Random Graphs While the Size Changes

While matching graphs and changing the density is quite interesting, we also want to know if the size of the graph has any role in the results. In this case study, we are going to match random graphs whose density is constant (we choose the density to be 0.5)

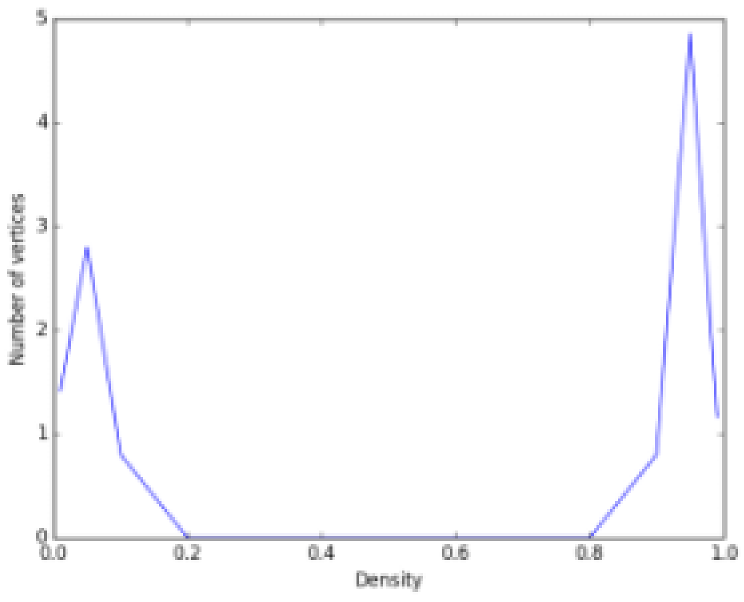


FIGURE 5.3: Standard deviation of the number of matches in a random graph

while we change the number of vertices in the graph. We will use graphs of size: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.

TABLE 5.8: Random Graphs: Mean and Standard deviation of the number of matched vertices ϵ.

Number of vertices	10	20	30	40	50	60	70	80	90	100
μ of the number of matched vertices	10	20	30	40	50	60	70	80	90	100
σ of the number of matched vertices	0	0	0	0	0	0	0	0	0	0

We see that in each case, we got a perfect isomorphism and so the standard deviation is always 0. In fact, the random graphs have this property that finding an isomorphism there is more easier than in general. In the next subsection, we are going to investigate into non-random graphs and see what results we get there.

5.2.3 Case Study 3: Matching Non-Random Graphs

On this section, we are going to test the matching algorithm in real graphs (non-random). All the graphs used in this section have been taken from [15]. In all graphs, it is extremely difficult to find an isomorphism, and the graphs have been chosen exactly for this reason. On each of the pair of graphs, we are going to do 25 Monte Carlo simulations and then see the results (as always, mean and standard deviation) on the number of matches we have found.

We are going to start by testing it on Coxeter graphs [11]. Coxeter graphs are relatively sparse graphs which contain 30 vertices and only 90 edges (density being 0.1). A picture of the graphs can be seen below:

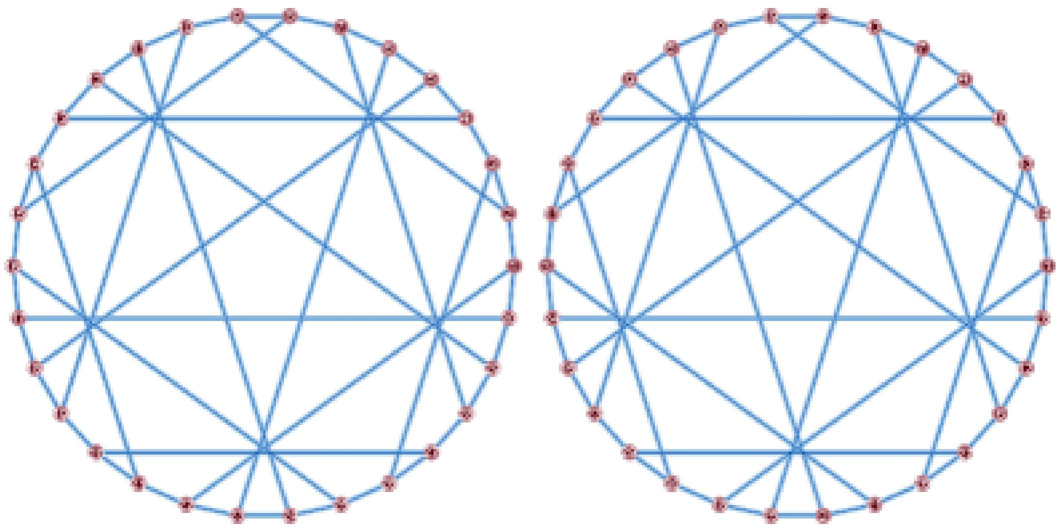


FIGURE 5.4: Coxeter graphs

After we perform the process of graph matching 25 times, we show the results in Figure 5.5.

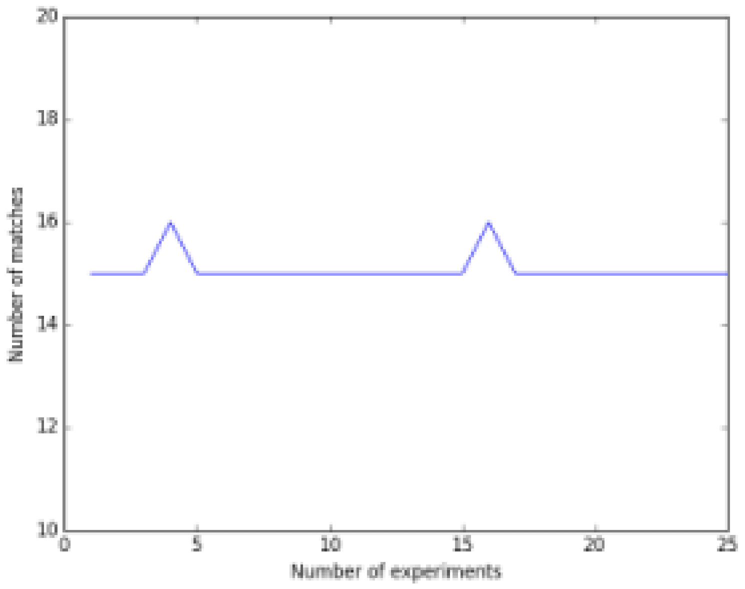


FIGURE 5.5: Matching Coxeter Graphs

We see that the results - while robust - are quite poor, and in 23 cases we have got just 15 matches (only half of the vertices), while in two cases we got 16 vertices. The mean of the number of matches is 15.08, while the standard deviation is 0.27.

Next, we are going to test the algorithm on similar, but not isomorphic graphs. We have chosen the Weisfeiler graphs [51], graphs that have 25 vertices and 300 edges (density is 0.48). The graphs can be seen in the figure below:

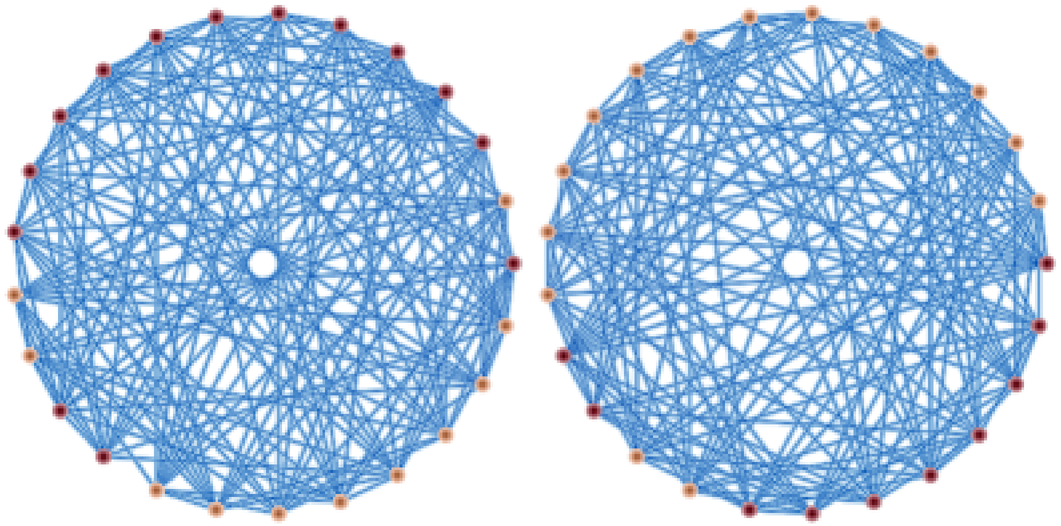


FIGURE 5.6: Weisfeiler graphs

We run the algorithm 25 times and we get the following results:

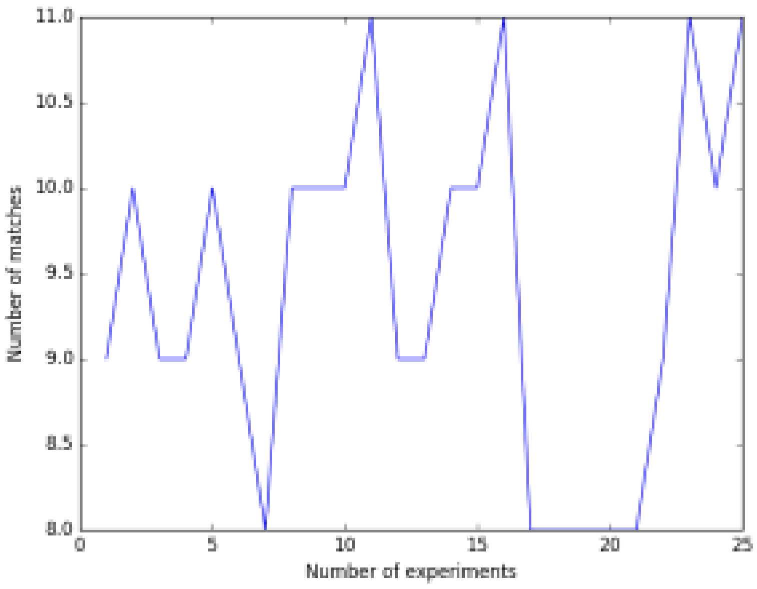


FIGURE 5.7: Matching Weisfeiler Graphs

We see that the number of matches varies from 8 to 11, with the mean being 9.4, and standard deviation being 1.01. The results are better than in the previous graph, which is to be expected considering that replicator dynamics don't work well on sparse graphs.

On the other hand, we observed that the algorithm works significantly better for smaller graphs. For example, taking the Petersen graphs [38] shown below (10 vertices, 30 edges, density being 0.3):

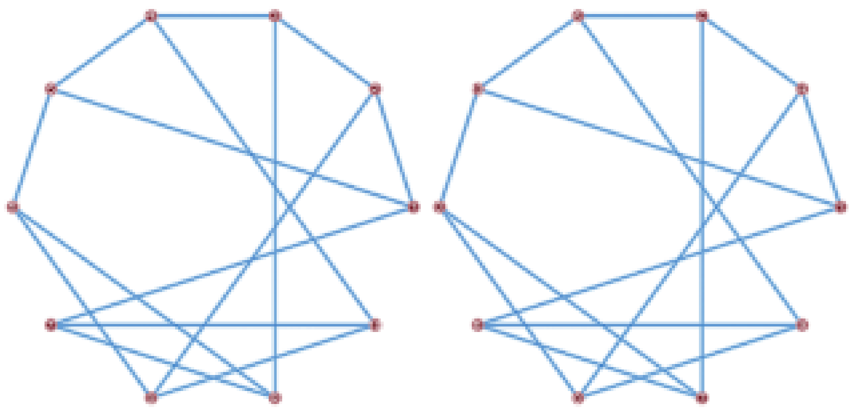


FIGURE 5.8: Petersen graphs

We test the matching algorithm by doing 25 simulations, and we get the following results:

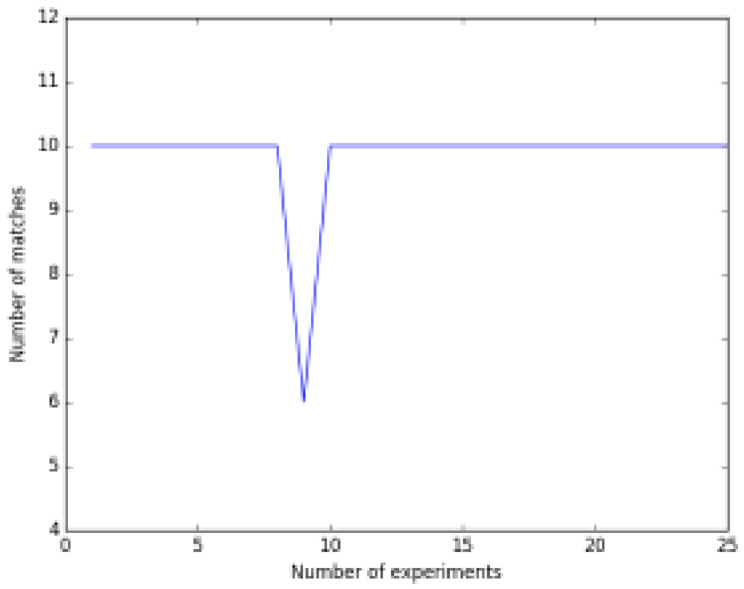


FIGURE 5.9: Matching Petersen Graphs

We see that in 24 out of 25 cases, we got a perfect isomorphism, while in one case we failed. The mean of the matches is 9.89, while the standard deviation is 0.78.

Because of this, and other similar results, when we will do the graph matching (after applying the Szemerédi lemma), we will try to perform the Szemerédi algorithm in that way that the graphs that are going to be matches, to not be that big.

We conclude this section by showing the evolution of the replicator dynamics in the last pair of graphs.

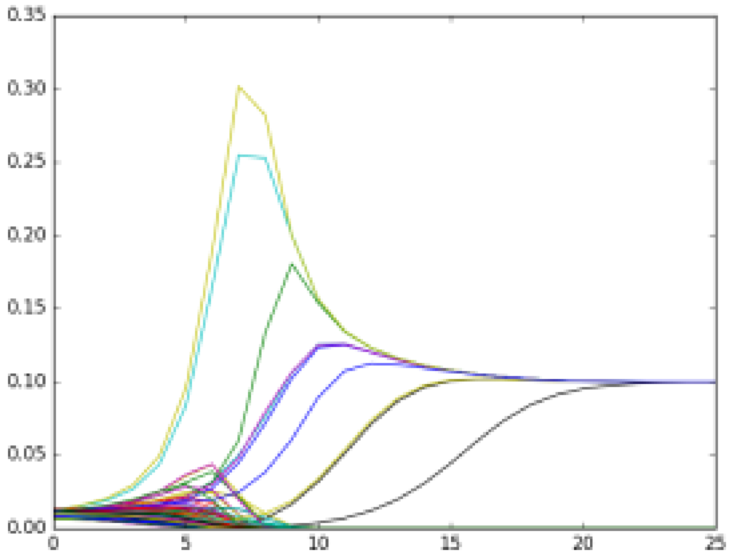


FIGURE 5.10: The evolution of replicator dynamics in Petersen graphs

We see how only after 10 iterations (we are using the exponential replicator dynamics with K = 5), half of the strategies have already converged, while after 25 iterations all 10 winning strategies have converged near the value 0.1, while the other 90 strategies have 'died', near the value 0.

5.3 Testing the Combined Algorithm

On this section, we are going to look at the bigger picture, and see how good the combined algorithm of regularity lemma - graph matching, performs in the context of matching large graphs. We will start by giving a review of the combined algorithm:

1) Having two graphs G_1 and G_2, apply the regularity lemma of Szemerédi, and so get two reduced graphs R_1 and R_2 - whom in theory - inherit the main properties of the

two original graphs. If possible, tune the parameters in that way which will produce reduced graphs of the same size.

2) The regularity lemma algorithm returns as output a list which shows in what class each element belongs, and a density matrix which shows the density between classes. There are a few possible approaches on how to match the reduced graphs. One way would be to threshold these two density matrices into ones and zeros, and then consider the matrices as unweighted graphs. We know that we can match the unweighted graphs by creating the association graph of them, and then finding a maximal clique on that graph (explained in details in Chapter 4). An another approach would be to skip the thresholding phase, and consider density matrices as weighted graphs. From chapter 4, we know that we can match weighted graphs by creating the association graph, and then finding a dominant set on it.

From empirical evidence, we prefer the second approach.

3) Now that we have matched the reduced graphs, we expand them into the original graphs, and then match the elements within the classes, considering only the intra-similarity.

Matching a lot of small graphs is more time/memory efficient, then matching just a pair of large graphs. In fact because of space constraints, matching a pair of large graphs might well be impossible (building the association graph would be impossible; of course, each entry of the association graph can be computed in isolation, which will solve the memory problems, but on the other side make the process of graph matching significantly slower because in that case we are going to call the function for a very high number of times).

While this combined algorithm makes possible matching larger graphs than before (and that at the fraction of needed time), there are some problems here. Mainly, the reduced graph is matched considering only the similarity between classes, while classes are matched based only on the intra-similarity between elements of the class. While theoretically speaking this is fine (reduced graph is the essence of the original graph, and so it inherits its properties), a lot depends on the quality of the partition. For ϵ too large, the quality of the partition might not be that good. Similarly, it is possible that on the partition there are a lot of irregular pairs. The second problem is that the regularity lemma tends to make the pairs be bipartite graphs (in fact, the algorithm considers the pairs to be bipartite) and that means that in many cases the elements in the class would be completely disconnected (no edges between them). We know that matching two bunches of vertices is trivial, which might mean that the final matching will suffer in the process.

So, we have to say that there is no theoretical guarantee that the matching is exact. In fact, we know that it is just 'an approximate matching'. For critical systems, or for graphs where we must have an exact matching, the algorithm (in the present stage) should not be used. On the other hand, where we need just an approximate matching (and there are many cases when we do, like in bioinformatics or graph indexing), or in cases where an exact matching is impossible (be it because of time or memory costs, then the algorithm might be used. Furthermore, the graph can be used when we need to do a simple coarse matching (match just regions of the graphs) or graph indexing, but if we need to do a precise fine matching, then the algorithm (in its current format) cannot be used.

In this section, we are going to test the combined algorithm on many datasets. We will start by testing it on UCI datasets, then test it on synthetic datasets, and finally test it on SNAP datasets. Then, we are going to do a detailed evaluation of the time needed for the algorithm, and in what parts of the algorithm the time is spent. That should give us a good view on the efficiency of the algorithm, and what are the trade-offs (if any) of having a small number of classes but with each class having a large number of vertices, or having a large number of classes with each class having a small number of vertices. Or maybe meeting somewhere in the middle where neither the number of classes, nor the number of vertices per each class is that high.

We have done the experiments in a laptop which has a core i7 processor, and 8GB of RAM memory. We will consider these results:

- the total time spent to perform the algorithm,

- the time spent to perform the regularity lemma (in both graphs),

- the time spent to create the association graph of the reduced graphs,

- the time spent to match the reduced graphs,

- the time spent to match the vertices in each class.

Obviously, the total time should equal the four other measures. On the tables below, we will give the results in the total number of seconds (in most cases, rounded), and as percentage of the total time.

For all the datasets (unless said otherwise), we did 10 Monte Carlo simulations.

5.3.1 Testing the Algorithm on the UCI Machine Learning Repository Datasets

We will start by testing the algorithm on a few databases taken by the UCI Machine Learning Repository. Because the data are given on feature-based format, we are going to use the kernel mentioned in section 5.1.1, with $\sigma = 1$. We will start by testing the algorithm on the Ionosphere dataset.

5.3.1.1 Ionosphere Dataset

Starting with the ionosphere dataset would be a natural choice, considering that we spent a lot of time working on this dataset, and in addition, in the previous few sections we tested both the regularity lemma and the graph isomorphism algorithm on it. The experiments done here would be quite simple. We will perform the combined algorithm, and then measure the number of matches (or more exactly the ratio of matches, which is a number between 0 - where we didn't manage to match any vertices - and 1 - where we managed to match all the vertices on the graphs), while also observing the number of classes, size of the class, and size of the exceptional size, and see if/what effect those variables have on the matching ration. As a general rule the ratio of matches cannot extend $\frac{number\ of\ vertices - size\ of\ the\ exceptional\ class}{number\ of\ vertices}$. Every vertex in the exceptional class, is a lost vertex, and we won't consider them (we know that while the classes can be considered as meaningful structures, the exceptional class doesn't represent a cohesive structure, and so cannot be used in the process of matching). This is the reason why we completely changed Heuristic 3, and with the change, now the size of the exceptional class is typically lower than using the previous heuristic (built by Sperotto and Pelillo [44] and Sarkozy et al [40]).

We give the results on the table below, for four values of epsilon. The values might look arbitrary, but in fact we tested the algorithm with different values of epsilon, and here we show just some of the results, acting as representatives.

TABLE 5.9: Ratio of matches in Ionosphere dataset ϵ.

ϵ	0.42	0.5	0.53	0.58
Number of classes	70	35	16	8
Size of class	5	10	21	43
Size of exceptional class	1	1	15	7
μ of ratio of matches	0.793	0.82	0.769	0.68
σ of ratio of matches	0.025	0.01	0.007	0.01

First observation is that the results are very robust (standard deviation is quite small). A more important observation though, is that it seems that a smaller ϵ typically gives better results. This is something that we were expecting and which makes perfect sense (smaller ϵ means a better partition). However, this doesn't happen always. As we see from the table, we got better results for $\epsilon = 0.5$ than for $\epsilon = 0.42$. While this might look strange, it makes sense from the results of the previous section (when we tested the matching algorithm). Where the datasets aren't random, the matching algorithm seems to work better for smaller graphs than for larger ones. Because for $\epsilon = 0.42$ we have twice as many classes as in the case of $\epsilon = 0.5$, the matching algorithm works better in the second case. Obviously, missing matches in the reduced graphs is more costly than missing matches in the original graph (each match we miss on the reduced graph is equivalent of missing 5 matches from the reduced graph, when $\epsilon = 0.42$). We also see that typically, the size of exceptional class increases with the increase of ϵ, although for all cases it stays relatively low (less than 4% of the size of the graph).

We will continue with the CPU time analysis. The table below gives those results we mentioned. As always we call the original graphs as G, while the reduced graphs as R. We call time t, and the association graph as W. We give the measurements as the total numbers of seconds, while in brackets we put as percentage levels.

We see that the best results are when neither the number of classes, nor the size of class isn't that large. In those cases, most of the time was spent on matching the reduced graphs, and then matching the nodes within classes. On the other side, when the number of classes was too large (and so the size of class being too small) most of the time was

Table 5.10: Time analysis for the Ionosphere dataset

ϵ	0.42	0.5	0.53	0.58
# of classes	70	35	16	8
Size of class	5	10	21	43
t spent regularity	22s (9.6%)	6s (5.88%)	2s (2.02%)	3.84s (0.85%)
t spent on W	96s (31.79%)	10s (9.8%)	1s (1.01%)	0.02s (0.0004%)
t spent matching R	182s (60.26%)	69s (67.65%)	41s (41.41%)	1.14s (0.25%)
t spent matching G	2s (0.66%)	17s (16.67%)	54 (54.54%)	445s (98.88%)
Total t spent	302s	102s	99s	450s

spent on creating the association graph W and then matching the reduced graphs. In the other hand, when the number of classes was small, but the size of class was large, almost the entire time was spent on matching the elements within classes. In all cases, the regularity lemma algorithm took less than 10% of the total time, while being completely negligible when the number of classes was small.

If we combine this information with the matching ratio, we see that the best ϵ for this dataset was $\epsilon = 0.5$ because not only it gave the best matching ratio, but also the entire algorithm run quite fast (in less than 2 minutes).

5.3.1.2 Haberman Dataset

We continue our experiments with the Haberman Dataset [23] from the UCI Machine Learning Repository. The dataset has 306 observations, each of them having 3 features which are given as integers. All observations belong in two distinct classes.

We will study the same response variables as in the previous dataset, and so the tables might look quite similar to each other.

Table 5.11: Ratio of matches in Haberman dataset ϵ.

ϵ	0.45	0.5	0.55	0.6
Number of classes	76	76	34	8
Size of class	4	4	9	38
Size of exceptional class	2	2	0	2
μ of ratio of matches	0.72	0.75	0.76	0.7
σ of ratio of matches	0.019	0.05	0.03	0.012

The results are very surprising. While we got the worst results for the largest ϵ, which we were expecting, in the other had we got worse results for $\epsilon = 0.45$ than for $\epsilon = 0.5$, despite the fact that both epsilons give the same number of classes. We would have expected that when different ϵ give the same number of classes, also the results to be similar to each other.

The best results we got were for $\epsilon = 0.55$, which is the second largest ϵ.

The table 5.12 for the time analysis, is given in the same format as before.

Table 5.12: Time analysis for the Haberman dataset

ϵ	0.45	0.5	0.55	0.6
# of classes	76	76	34	8
Size of class	4	4	9	38
t spent regularity	6s (1.4%)	7s (1.08%)	1.7s (4.47%)	0.7s (0.29%)
t spent on W	142s (33.18%)	337s (51.93%)	0.1s (0.26%)	0.1s (0.004%)
t spent matching R	278s (64.95%)	302s (46.53%)	11.2s (29.47%)	0.2s (0.08%)
t spent matching G	2s (0.47%)	3s (0.46%)	25 (65.79%)	244s (99.59%)
Total t spent	428s	649s	38s	245s

The same pattern as before can be seen here. Best performance is on the case when neither the number of classes, nor the size of class is that large. Unlike in the previous dataset, we have to choose here if we want time efficiency or having a good matching rate. The best matching rate happened when $\epsilon = 0.5$, and that was around 6% higher than when $\epsilon = 0.55\%$. However, there is needed approximately 20 times more time to perform the algorithm for that parameter.

Like in the previous example, when the number of classes was large, most of the time was spent on building the association graph and matching the reduced graphs, while when the size of classes was large, most of the time was spent on matching elements of the classes. In all cases, the regularity lemma was performed very fast, never needing more than 5% of the total time, which was actually an outlier considering that in the other 3 cases it needed just around 1% of the time spent on the entire algorithm.

Table 5.13: Ratio of matches in Red Wine dataset ϵ.

ϵ	0.35	0.4	0.43	0.46
Number of classes	133	84	66	66
Size of class	12	19	24	24
Size of exceptional class	3	3	15	15
μ of ratio of matches	0.79	0.81	0.78	0.77
σ of ratio of matches	0.019	0.025	0.027	0.008

5.3.1.3 Red Wine Dataset

On this section, we are going to repeat the experiment on the Red Wine Dataset. Red Wine Dataset is part of the Wine Quality Dataset [10] from UCI Machine Learning Repository. It contains 1599 observations, each of them having 12 attributes (features). It is by far the largest UCI dataset we have used for our experiments, and because of the time constraints, we have done 5 instead of 10 simulations.

The table 5.13 shows the results of the response variables:

We see that the results are extremely robust. Not only that the standard deviation is very small for each of the epsilons, but it looks like that even the changes in ϵ don't change much the final results. Saying that, the pattern is as before, with small ϵ typically giving better results than large ϵ, though the difference isn't that big.

Using the same approach in the Red Wine dataset, we get the following CPU time results:

Table 5.14: Time analysis for the Red Wine dataset

ϵ	0.35	0.4	0.45
# of classes	133	66	66
Size of class	12	24	24
t spent regularity	356s (8.77%)	177s (20.49%)	205s (22.6%)
t spent on W	3702s (91.23%)	129s (14.93%)	138s (15.21%)
t spent matching R	68s (1.68%)	212 (24.54%)	213s (23.48%)
t spent matching G	75s (1.85%)	346s (40.05%)	351s (38.7%)
Total t spent	4058s	864s	907s

We see that we got best results for $\epsilon = 0.4$ which also gives the best results when it comes to matching ratio. In this case, most of the time is spent on doing the matching of elements within a class, while the other three measures take around 1/3 of the time. For $\epsilon = 0.35$ we see that the algorithm took a very long time to perform (more than an hour), with the vast majority of this time (more than 90%) was needed to create the association graph. That is because the number of elements in the association graph is $O(n^4)$ compared with the number of vertices. So, with 133 classes, the adjacency matrix of the association graph has 312900721 entries. For $\epsilon = 0.45$, the time spent on the algorithm is around the same as for $\epsilon = 0.4$.

It needs to be said that this time, the Szemerédi Regularity Lemma algorithm took more time in average than in the previous experiments, in two cases taking more than 20% of the total time spent on the combined algorithm.

Considering that $\epsilon = 0.4$ seems to give the best results both in efficiency and in the isomorphism rate, this might be the recommended ϵ for this dataset.

With this experiment, we conclude our testing in quality for the UCI Machine Learning Repository Datasets.

5.3.2 Testing the Algorithm on Randomly Generated Datasets

On this section, we are going to test the algorithm on randomly generated datasets. We will use the same platform as in section 5.1.2 (expand the graph in Figure 5.1). The graph is more sparse than the graphs we used in the previous section, having density only around 10%.

Like always we will give the results of mean and standard deviation, and see how the precision of the algorithm is affected from the other variables. However, this time we will also use different values for parameter l (the parameter which defines in how many classes each class will get separated).

Table 5.15: Ratio of matches in a random dataset for l = 2

Number of classes	16	38	80
Size of class	50	21	10
Size of exceptional class	0	2	0
μ of ratio of matches	0.81	0.918	0.952
σ of ratio of matches	0.008	0.008	0.011

The results seem significantly better than in UCI datasets, and the results are as expected. The more classes we get, the better seem to be the results of the matching. Indeed, this is what should normally happen as we can see from the results, we get excellent matching when we have 80 classes (10:1 compression ratio), but not that good matching when we have only 16 classes (50:1 compression ratio). For 38 classes (21:1 compression ratio) the results are on the middle. This is the first time we see that in all cases the results seem to follow the line of thought, that more classes (less compression) means better matching ratio.

Now, we will look at the results if we have l = 3.

TABLE 5.16: Ratio of matches in a random dataset for l = 3

Number of classes	20	61
Size of class	39	13
Size of exceptional class	20	7
μ of ratio of matches	0.789	0.87
σ of ratio of matches	0.023	0.185

We see immediately that the same thing as in the previous experiment has happened, more classes means a better matching ration. However, we also see that the results are slightly worse than when we did the experiment with l = 2. One reason for that might be because the size of the exceptional size is slightly larger than before.

In fact, if we combine the tables and sort them in increasing order, we see:

TABLE 5.17: Ratio of matches in a random dataset

Number of classes	16	**20**	38	**61**	80
Size of class	50	**39**	21	**13**	10
Size of exceptional class	0	**20**	2	**7**	0
μ of ratio of matches	0.81	**0.789**	0.918	**0.87**	0.952
σ of ratio of matches	0.008	**0.023**	0.008	**0.185**	0.011

The bold ones are those when $l = 3$.

We see that even in cases when the number of classes (for $l = 3$) is 'slightly' higher than the number of classes (for $l = 2$), we get better results for $l = 2$. For example, the results are for around 3% worse when we have 20 classes - but $l = 3$ - than when we have 16 classes, but $l = 2$. Similarly, the results are 4% worse when we have 61 classes (and

$l = 3$) than when we have 38 classes (and $l = 2$). We can only suppose that the same pattern would have continued if we did an experiment for $l = 3$ which resulted with a higher number of classes than 80.

We can conclude that at least for this graph, parameter l has an important role for the graph matching.

Looking at the time performance.

TABLE 5.18: Time analysis for a random dataset

# of classes	16	38	80
Size of class	50	21	10
t spent regularity	114s (6.76%)	23s (6.15%)	28s (8.3%)
t spent on W	1s (0.06%)	2s (0.53%)	104s (30.86%)
t spent matching R	24s (1.42%)	39s (10.43%)	160s (47.48%)
t spent matching G	1649s (97.75%)	310s (82.88%)	45s (13.35%)
Total t spent	1687s	374s	337s

We got the best results, when the number of classes was the largest, which goes against other results (generally, we expect the algorithm to run fastest when neither the size, nor the number of classes is that big). On the other side, we got the best results when the number of classes was low, but the size of a class was quite high. We see that the Szemerédi algorithm didn't need much time to be performed in any of the cases, while bar in the last case (when the number of classes was very high), the absolute majority of time was spent on matching the original graphs (fine matching).

Combining all things, we see that the best case is when the number of classes is large, because not only it gives the best results, but also the algorithm runs faster than in any other case.

5.3.3 Testing the Algorithm on SNAP Datasets

The final experiment in this part, would be on a SNAP graph. We won't use the same graph as in section 5.1.3 because it would have been hard to process that graph. Instead, we will use a similar graph which is around half its size and which we took from the same dataset [31]. The graph describes General Relativity and Quantum Cosmology collaboration network from the e-print arXiv. The graph has 5242 nodes with 14496 vertices (3 vertices per node), so the density of the graph is just 0.5%, making it extremely sparse. Some interesting facts about the graph are that the largest strongly connected component has 4158 nodes and 13428 edges, making the graph a well connected graph. The diameter of the graph is 17.

Because the graph is very large (in comparison with the other graphs we studied), we needed to do some manual tuning in the algorithm, and in the end we are going to give the results only for a single value of parameter ϵ.

We did 5 experiments, and in the table 5.19, we give the results of them.

TABLE 5.19: Ratio of matches in a SNAP dataset ϵ.

ϵ	0.07
Number of classes	131
Size of class	40
Size of exceptional class	2
μ of ratio of matches	0.952
σ of ratio of matches	0.006

The results are fantastic, which was kind of surprising. Matching the original graph would have been extremely difficult, not only because of its size, but also because replicator dynamics do a terrible job in sparse graph. However, the combined algorithm seem to have overcome that problem quite well.

For the time analysis, we get the results in Table 5.20.

TABLE 5.20: Time analysis for a SNAP dataset

# of classes	131
Size of class	40
t spent regularity	1409s (7.79%)
t spent on W	1862s (10.31%)
t spent matching R	4556s (25.2%)
t spent matching G	10250s (56.7%)
Total t spent	18077s

We don't have much data for this experiment (we gave the results in only one ϵ), but we see that the algorithm of the regularity lemma was performed quite fast (the usual less than 10% of the total time), with the majority of time spent on fine matching, while also a significant portion of time being spent on matching the reduced graph (which has 130 vertices). The total time for the algorithm needed to be performed was around 5 hours, which isn't that much considering that matching two uncompressed graphs of those sizes, is virtually impossible for a normal computer, and challenging even for a cluster of computers.

With this experiment, we finish this evaluation of the algorithm.

5.4 Discussion

This chapter is arguably the most important work on this thesis, and here we combined all things that we discussed in the previous chapters. We did some experiments, starting from evaluating the regularity lemma, evaluating the matching algorithm, and finally evaluating the combined algorithm.

We saw that the regularity lemma is generally performed very fast, and it gives quality results. On the other hand, we saw that for some types of graphs, the matching algorithm might struggle to give good results, and at times even fail. More importantly, we observed that in all types of datasets, the matching algorithm did a very good job when we did the coarse matching (matching just regions of graphs, regions defined by the regularity lemma), but the results on the fine matching (matching the entire graphs), are more difficult to interpret. While the results looks good (around 80% on UCI Machine Learning Repository Datasets), and around 95% on the random graphs and SNAP datasets, the possibility for false positives is quite high. In particular, false positives are present when the small graphs are bipartite graphs, in which case, the isomorphism on those graphs becomes trivial. Considering that matching the reduced graphs (coarse matching) takes into account only the regular partition, but matching the classes takes into account only the intra-similarity, it is difficult to know - at this stage - how good the results really are.

We did a 'CPU time analysis', and we saw that the algorithm - for most part - gives best results when neither the size of the classes, nor the number of classes is too high. Considering that when the number of classes is too low, the size of the class becomes too high (and vice versa), then the road to go is to have both those quantities in an acceptable range. This would mean to have parameter ϵ not too high (because in that case, the size of classes would be quite high) and not too low (because in that case the number of classes will grow large). In all cases, the Szemerédi's Regularity Lemma Algorithm was performed relatively fast compared with the time spent on the entire algorithm.

We also saw that in most cases (SNAP dataset being an exception), we got the best matching when neither the number of classes, nor the size of classes is high, which coincides with the time analysis results. So, we are lucky in that aspect considering that we are getting best matching results while we spent less time in the algorithm.

The recommendation from this chapter is that we need to find ϵ parameters that give a moderate number of classes, and a moderate size of classes, to get the best results in both time and performance. The results were quite robust, and weren't affected that much from the structure of the graphs (graphs being sparse or not).

Chapter 6

Conclusions and Future Work

As the final chapter of this thesis, here we are going to review the work done in the previous chapters, make conclusions, and finish the thesis by mentioning possible ideas that can improve and extend the algorithm.

6.1 Review

The goal of the thesis was to study the application of the regularity lemma of Endre Szemerédi, especially in the context of graph matching and graph isomorphism. We started the thesis by giving the story behind the regularity lemma, and mentioning how what was originally just a tool on proving the now-called Szemerédi's theorem, eventually managed to become the central result on the extremal graph theory. Szemerédi himself won the most prestigious price in mathematics, the Abel price, while Timothy Gowers and Terence Tao won field medals, partially for the work done in the Szemerédi's theorem and Szemerédi's lemma.

However, the lemma being non-constructive in nature meant that its usefulness was very limited outside of pure mathematics. For more than 15 years, there was no algorithm that showed how the lemma can be used in practice. Things changed a bit when a combined group of Israeli, American and German researchers lead by Noga Alon (winner of Erdos prize and Godel prize), eventually managed to develop an algorithm. Despite that the algorithm has a sub-cubic complexity, its usefulness in practice (in pure form) is nonexistent because the suppressed constants in the big-Oh notations are incredibly big. In fact, in literature often it has been said that the algorithm has astronomical complexity. Furthermore, Timothy Gowers has mathematically proven that this complexity is necessary. Other algorithms were developed (like the Frieze-Kannan one) in the later years, but the problem remained.

The first documented application of the regularity lemma (that we know of) was done in a thesis of Anna Sperotto, here at Ca' Foscari. That work later resulted in a paper from Sperotto and Pelillo, where they successfully used the regularity lemma as a pre-processing step in graph clustering (using the dominant set algorithm). Something similar was done in a thesis of Shubhendu Trivedi (Worcester Polytechnic Institute), which eventually resulted in a paper from Sarkozy, Song, Szemerédi and Trivedi, where they used the algorithm almost in the same way, but this time using the spectral clustering algorithm. Both teams developed heuristics which made possible for the lemma to be used in practice, but on the other side removed the guarantee that the graph will eventually converge to a regular partition.

The second central topic of this thesis is that of graph isomorphism and graph matching. While the problem of graph isomorphism is still open, no-one has managed to find a polynomial algorithm for it. During the time we were working on this thesis, Laszlo Babai claimed that he has developed an algorithm that does graph isomorphism on quasi-polynomial time. There is no paper yet which gives the procedure, but professor Babai has put online a lecture where he shows the procedure. Still, the algorithm in practice won't be more efficient than many heuristic algorithms which have been developed over the years. One such algorithm, is the one which transforms the problem of graph isomorphism into the problem of finding the maximal clique in the association graph. While the maximal clique problem is NP-hard, there are heuristic algorithms which solve the problem reasonably fast. One such algorithm is the replicator dynamics one, which is a game theoretical approach. The algorithm gets easily generalized by using dominant sets instead of maximal cliques.

The idea of the thesis was to combine these two algorithms, in order to find an efficient algorithm which does graph matching. We had set two goals:

1) Develop an efficient algorithm which does coarse graph matching (matches regions of the graph).

2) Develop an efficient algorithm which does approximate fine matching.

The coarse matching part was relatively straightforward. We used the regularity lemma to compress the graphs, knowing that each vertex in the compressed (reduced) graphs, represent a class (or region) of the original graphs. By matching the compressed graphs, we are matching regions of the original graphs.

For the second problem, we continued matching the elements in classes of graphs based on the class structure. We admit that this is just an approximate matching, because this doesn't consider the edges which go to other classes. While that part of the problem has been covered by the regularity algorithm, that doesn't prevent false positives, especially in cases where the classes are just a bunch of vertices (without any edge between them), or fully connected graphs.

6.2 Conclusions

We implemented the work mentioned in the thesis using Python 2.7 and a few external libraries (numpy, pandas, networkx). Because we need the class of the exceptional class to be very small, we needed to completely change one of the heuristics developed from previous works.

We needed to make some decisions on how to match the reduced graphs. Considering that the result from the regularity lemma is a weighted graph (edges being the density between classes of the graph) we had to decide on either using some threshold in edges, and then use maximal clique algorithm, or use dominant sets in the weighted graphs. In addition, we needed to decide if we were going to consider also the non-regular pairs. We saw that the best results were when we used dominant sets approach in addition to considering non-regular pairs.

The results are partially satisfactory. We think that we have successfully solved the problem of coarse graph matching which was the first goal of the thesis. And while we have managed to do approximate fine matching, we aren't sure on the correctness of the results. At this moment, the algorithm cannot be used to do precise matching on graphs, but can be used if we need only an approximate matching in addition to matching classes of the graphs.

We did experiments in three types of datasets. The first type of datasets were taken from UCI Machine Learning Repository, and were feature-based datasets. We used a similarity kernel to transform those datasets into similarity-based datasets, and then using some threshold, we defined the edges between the vertices. The second type were randomly generated datasets, while the third type were real word datasets taken from Stanford Large Network Dataset Collection. UCI datasets generated dense graphs, while the randomly generated datasets were relatively sparse graphs. SNAP datasets were extremely sparse graphs.

In all three types of datasets we got very good results on matching regions, while on approximate matching, the results for UCI datasets were slightly higher than eighty pertcen, while for the other two types of datasets were around ninety-five percent. We aren't sure if this is because of the nature of datasets, or because the algorithm performs better in sparse graphs.

6.3 Future Work

Obviously, the big elephant in the room is going from coarse matching to fine matching, while reducing the number of false positives. The idea of using intra structure of the classes is a nice start, but we need a more intelligent way of doing it. Ideally, we need an algorithm that considers at the same time both class matching and intra structure of the classes. While we implemented a heuristic in regularity lemma that considers also the intra-similarity, that doesn't solve the cases where in classes we have just a bunch of vertices without any edge, or where the classes are completely connected.

We are also interested on improving the regularity lemma algorithm. We are going to investigate on the quality of certificates (witnesses) when we partition classes. Similarly to other existing algorithms, we randomly choose certificates to do the partition. We feel that there are more intelligent ways of doing so, like taking into account the measure of irregularity the certificates have, or the size of certificates.

We are planning to see how the algorithm will work if we change the isomorphism/-matching part of the algorithm. Especially is Babai's claim results with an efficient result algorithm, which can run in around the same time as the heuristic algorithms, we would be very interested to see how much that would improve the combined algorithm. In any case, we plan to use different types of heuristic algorithms used for graph matching and see the results.

An another thing we would like to address in the future, is to see in what other fields we can use the regularity lemma. As far as we know, in the past it has been used only in graph clustering, and now we're using it on graph matching, but we feel that it can be used for many other topics. Considering that the reduced graphs have the same properties as the original graphs, the lemma has great potential to be used in all types of graph problems.

Finally, the theoretical knowledge on the lemma (or the lack of it), has always been problematic, and by using heuristics, the lack of information on the nature of the lemma increases. We hope that one day, this problem will be addressed and so we would have a better understanding on the nature of this powerful tool.

Bibliography

[1] Alon, N., Duke, R. A., Lefmann, H., Rodl, V., and Yuster, R. (1994). *The Algorithmic Aspects of the Regularity Lemma.* J. Alg., 16.

[2] Asuncion, A. and Newman, D. (2007). *UCI Machine Learning Repository.* http://archive.ics.uci.edu/ml/index.html.

[3] Babai, L. (2015). *A Quasipolynomial Time Algorithm for Graph Isomorphism: The Details.* http://people.cs.uchicago.edu/ laci/2015-11-10talk.mp4.

[4] Baird, H. S. and Cho, Y. E. (1975). *An artwork design verification system.* Proceedings of the 12th Design Automation Conference.

[5] Barrow, H. G. and Burstall, R. M. (1976). *Subgraph isomorphism, matching relational structures and maximal cliques.* Inform. Process. Lett.

[6] Bomze, I. M., Budinich, M., Pardalos, P. M., and Pelillo, M. (1999). *The maximum clique problem.* Handbook of combinatorial optimization.

[7] Bron, C. and Kerbosch, J. (1973). *Algorithm 457: finding all cliques of an undirected graph.* Communications of the ACM.

[8] Condon, A. and Karp, R. M. (2001). *Algorithms for graph partitioning on the planted partition model.* Random Structures & Algorithms.

[9] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms.* MIT Press.

[10] Cortez, P., Cerdeira, A., Almeida, F., Matos, T., and Reis, J. (2009). *Modeling wine preferences by data mining from physicochemical properties.* In Decision Support Systems, Elsevier, 47(4):547-553.

[11] Coxeter, H. and Tutte, W. (1958). *The Chords of the Non-Ruled Quadratic.* PG(3,3), Canad. J. Math.

[12] Crow, J. F. and Kimura, M. (1970). *An introduction to population genetics theory.* New York: Harper Row.

[13] Czygrinow, A. and Rodl, V. (2000). *An algorithmic regularity lemma for hypergraphs.* SIAM J. Comput.

[14] Deskus, C. (2011). *An Introduction to the Regularity Lemma.*

[15] Dharwadker, A. and Tevet, J.-T. (2009). *The Graph Isomorphism Algorithm.* Procedings of the Institute of Mathematics and the Structure Semiotics Research Group.

[16] Diestel, R. (2010). *Graph Theory.* Graduate Texts in Mathematics, Volume 173, Springer-Verlag, Heidelber.

[17] Erdos, P. and Turan, P. (1936). *On Some Sequences of Integers.* J. London Math, Soc, 11.

[18] Frieze, A. and Kannan, R. (1999). *A Simple Algorithm for Constructing Szemerédi's Regularity Partition.* Electrong. J. Comb, 6 (1).

[19] Garey, M. R. and Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness.* San Francisco: W. H. Freeman.

[20] Gibbons, L. E., Hearn, D. W., Pardalos, P. M., and Ramana, M. V. (1997). *Continuous characterization of the maximum clique problem.* Math. Oper. Res.

[21] Golub, G. and Loan, C. V. (1989). *Matrix Computations.* John Hopkins University Press, London.

[22] Gowers, W. T. (1997). *Lower bounds of tower type for Szemeredis uniformly lemma.* Geom. Funct. Anal.

[23] Haberman, S. J. (1976). *Generalized Residuals for Log-Linear Models.* Proceedings of the 9th International Biometrics Conference, Boston.

[24] Hofbauer, J. and Weibull, J. W. (1996). *Evolutionary selection against dominated strategies.* J. Econ. Theory.

[25] Kohayakawa, Y. (1997). *Szemerdi's regularity lemma for sparse graphs.* Springer-Verlag.

[26] Kohayakawa, Y., Rodl, V., and Thoma, L. (2003). *An optimal algorithm for checking regularity.* SIAM J. Comput.

[27] Komlos, J., Shokoufandeh, A., Simonovitz, M., and Szemerédi, E. (2002). *The regularity lemma and its applications in graph theory.* Theoretical Aspects of Computer Science: Advanced Lectures, Springer.

[28] Komlos, J. and Simonovitz, M. (1996). *Szemerédi's Regularity Lemma and its applications in graph theory.* Combinatorics, Paul Erdos in Eighty.

[29] Komls, J., Sarkozy, G., and Szemerdi, E. (1997). *Blow-up lemma.* Combinatorica, Springer.

[30] Kozen, D. (1978). *A clique problem equivalent to graph isomorphism.* SIGACT News.

[31] Leskovec, J., Kleinberg, J., and Faloutsos, C. (2007). *Graph Evolution: Densification and Shrinking Diameters.* ACM Transactions on Knowledge Discovery from Data (ACM TKDD).

[32] Leskovec, J. and Krevl, A. (2011). *SNAP Datasets: Stanford Large Network Dataset Collection.*

[33] Motzkin, T. S. and Straus, E. G. (1965). *Maxima for graphs and a new proof of a theorem of Turan.* Canad. J. Math.

[34] Pardalos, P. M. and Phillips, A. T. (1990). *A global optimization approach for solving the maximum clique problem.* Int. J. Comptuter Math.

[35] Pavan, M. and Pelillo, M. (2003). *A new graph-theoretic approach to clustering and segmentation.* CVPR.

[36] Pelillo, M. (1999). *Replicator Equations, Maximal Cliques, and Graph Isomorphism.* Neural Computation.

[37] Pelillo, M. and Jagota, A. (1995). *Feasible and infeasible maxima in a quadratic program for maximum clique.* J. Artif. Neural Networks.

[38] Petersen, J. (1891). *Die Theorie der regularen Graphen.* Acta Math.

[39] Rodl, V., Nagle, B., Skokan, J., Schacht, M., and Kohayakawa, Y. (2005). *The hypergraph regularity method and its applications.* PNAS.

[40] Sarkozy, G., Song, F., Szemerédi, E., and Trivedi, S. (2012). *A Practical Regularity Partitioning Algorithm and its Applications in Clustering.* arXiv preprint arXiv:1209.6540.

[41] Schmidt, E. (2010). Techonomy Conference.

[42] Sigillito, V. (1989). *UCI Machine Learning Repository, Ionosphere Dataset.* https://archive.ics.uci.edu/ml/datasets/Ionosphere, Johns Hopkins University.

[43] Smith, J. M. (1982). *Evolution and the theory of games.* Cambridge University Press.

[44] Sperotto, A. and Pelillo, M. (2007). *Szemerédi's Regularity Lemma and its Applications to Pairwise Clustering and Segmentation.* EMMCVPR, LNCS 4679., Springer.

[45] Szemerédi, E. (1975). *On Sets of Integers Containing no k Elements in Arithmetic Progression.* Acta Arithmetica 27.

[46] Szemerédi, E. (1978). *Regular Partitions of Graph.* Proc. Colloque Inter.

[47] Taylor, P. and Jonker, L. (1978). *Evolutionary stable strategies and game dynamics.* Math. Biosci.

[48] Torsello, A., Bulo, S. R., and Pelillo, M. (2006). *Grouping with asymmetric affinities: A game-theoretic perspective.* Proc. CVPR.

[49] Trevisan, L. (2006). *Szemerédi's Theorem.* Blog of professor Trevisan in 'In Theory'.

[50] Weibull, J. W. (1995). *Evolutionary game theory.* Cambridge MA, MIT Press.

[51] Weisfeiler, B. (1976). *On Construction and Identification of Graphs.* Springer Lecture Notes Math.