DISTANCE BASED CANONICAL LABELING ALGORITHMS WITH APPLICATIONS TO GRAPH MATCHING

Pamela Tabak

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Daniel Ratton Figueiredo

Rio de Janeiro
Março de 2020

DISTANCE BASED CANONICAL LABELING ALGORITHMS WITH APPLICATIONS TO GRAPH MATCHING

Pamela Tabak

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientador: Daniel Ratton Figueiredo

Aprovada por: Prof. Daniel Ratton Figueiredo
Prof. Jayme Luiz Szwarcfiter
Prof. Fábio Protti

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2020

Tabak, Pamela

Distance Based Canonical Labeling Algorithms with Applications to Graph Matching/Pamela Tabak. – Rio de Janeiro: UFRJ/COPPE, 2020.

XII, 88 p.: il.; 29, 7cm.

Orientador: Daniel Ratton Figueiredo

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2020.

Referências Bibliográficas: p. 85 – 88.

1. canonical labeling. 2. graph matching. 3. graph isomorphism. I. Ratton Figueiredo, Daniel. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

Agradeço aos meus pais, Claudio e Lilian, que sempre me apoiaram e me deram todas as oportunidades de estudar. O amor incondicional e dedicação dos dois me fez crescer em um ambiente onde tudo é possível, desde que nos esforcemos para conquistar nossos objetivos.

Agradeço a todos os meus amigos, pessoas com quem eu puder contar e confiar em todos os momentos e me fizeram ter lembranças e histórias maravilhosas para contar. Em especial, ao meu melhor amigo e companheiro, Eric Reis, que me apoiou ao longo deste projeto, assim como em todas as minhas aventuras nesses últimos anos. Também preciso agradecer aos meus dois maiores companheiros caninos, Brad e Martini, que estiveram presentes ao meu lado durante toda a produção desta dissertação.

À Universidade Federal do Rio De Janeiro sou grata por ter passado os últimos sete anos estudando nela, desde o início da minha jornada na graduação em Engenharia da Computação e Informação até o fim do meu mestrado. Aos professores do curso, agradeço por todas as aulas e ajuda oferecida, dentro e fora de sala, que me deram as ferramentas necessárias para me tornar mestre em Engenharia de Sistemas e Computação. Em especial, agradeço ao meu orientador, Daniel Ratton Figueiredo, que me acompanhou desde o meu projeto final de graduação, dedicando seu tempo para ensinar muito mais do que poderia imaginar.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ROTULAÇÃO CANÔNICA BASEADA EM DISTÂNCIAS COM APLICAÇÕES PARA O PROBLEMA DE EMPARELHAMENTO DE GRAFOS

Pamela Tabak

Março/2020

Orientador: Daniel Ratton Figueiredo

Programa: Engenharia de Sistemas e Computação

Grafos são estruturas matemáticas que codificam relações entre pares de objetos, que são normalmente identificados por rótulos. Entretanto, grafos podem representar relacionamentos entre qualquer tipo de objetos e um problema fundamental é determinar se dois grafos são estruturalmente equivalentes. Este problema clássico da teoria dos grafos, chamado isomorfismo de grafos, pode ser atacado com algoritmos de rotulação canônica, que rotulam os vértices de um grafo se baseando completamente na estrutura do grafo e independente da rotulação inicial dos vértices. Uma pergunta mais recente é sobre como alinhar os vértices de dois grafos estruturalmente similares, um problema conhecido como emparelhamento de grafos. Esta dissertação aborda esse problema com algoritmos de rotulação canônica. Em particular, propõe uma nova abordagem baseada apenas em distâncias, que resolve o problema de emparelhamento de grafos sob algumas condições enquanto sempre resolve o problema de isomorfismo em grafos. Duas variações são consideradas e ambas são implementadas e avaliadas em modelos de grafos aleatórios e redes reais, sob um modelo simples de remoção de arestas. Os algoritmos clássicos de rotulação canônica também são avaliados e comparados com a abordagem proposta baseada em distâncias, que tende a ser superior no alinhamento de grafos semelhantes.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

DISTANCE BASED CANONICAL LABELING ALGORITHMS WITH APPLICATIONS TO GRAPH MATCHING

Pamela Tabak

March/2020

Advisor: Daniel Ratton Figueiredo

Department: Systems Engineering and Computer Science

Graphs are mathematical structures that encode pairwise relationships between objects, which are usually identified with labels. However, graphs can represent relationships between any kind of objects and a fundamental problem is to determine if two graphs are equivalent from a structural point of view. This classic problem from graph theory, called graph isomorphism, can be tackled by canonical labeling algorithms, that label the graph nodes completely based on the graph structure and independent of the nodes' original labels. A more recent question is on how to align the nodes of two graphs that have a similar structure, a problem known as graph matching. This dissertation approaches this problem with canonical labeling algorithms. In particular, it proposes a novel approach based solely on distances, that solve the graph matching problem under some conditions while always solving graph isomorphism. Two variations are considered and both are implemented and evaluated on random graph models and real networks, under a simple edge removal model. Classic canonical labeling algorithms are also evaluated and compared to the proposed distance-based approach, which tends to be superior in aligning two similar graphs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Graphs are mathematical structures used to model relations between objects and can be applied to different problems, like the travel salesman problem. Graphs are composed of nodes and edges that encode the relationships between the corresponding pair of objects. The nodes are usually labeled uniquely, which allows for their identification. However, the meaning of a node and its respective label is arbitrary, allowing graphs to be applied to any context. For example, in the social network induced by friendship on Facebook, where nodes represent people and edges represent a friendship. Also, in the protein-protein network, that represents proteins, as nodes, and the interaction between them, as edges. Example of both graphs described are shown in Figure 1.1.

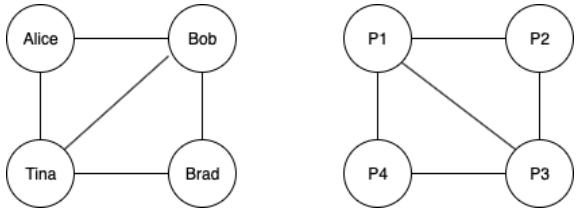


Figure 1.1: Example of two graphs representing different purposes.

An interesting question is if two graphs where nodes have different labels are equivalent from a structural point of view. Looking at the graphs shown in Figure 1.1, although they represent different objects, it is easy to see that their structure is identical. In particular, if labels are removed from both graphs, we have two unlabelled identical graphs. In fact, determining if two graphs are structurally identical is known as the graph isomorphism problem, a classic and well-studied problem in graph theory and computer science. While determining the isomorphism of graphs in Figure 1.1 is trivial, this problem is not known to be solvable in polynomial time in arbitrary graphs, and it is also not known to be NP-complete [3].

One approach to tackle the graph isomorphism problem is to label the graph nodes in a way that is completely based on the graph structure, independent of the

node's original labels. This is known as canonical labeling [4–8]. For special classes of graphs, canonical labeling algorithms can often solve the isomorphism problem efficiently [4, 9].

1.1 Graph Isomorphism Applications

Graphs are used by different areas to encode various kinds of relational information. For example, social network relationships encode relationships between people, biological networks encode different relationships between organisms, where it is possible to represent a metabolic network where the nodes represent genes and edges represent the interactions between them [10], revolutionizing the way of analyzing and studying biology. This broad interest in graphs has lead to the emergence of network science, an academic discipline with a focus on studying real networks, including their structure. The graph isomorphism problem appears in many of these different contexts, given the importance of determining if two graphs are structurally identical.

A variation of the isomorphism problem is to determine if a given graph is isomorphic to a subgraph of a larger graph. For example, determining if a clique with k nodes is a subgraph of a larger graph. This technique is often used to solve different problems, like pattern discovery in databases and bioinformatics [11].

Another application of isomorphism is related to privacy and data disclosure. Data privacy concerns are constantly increasing, especially after the European Union issued the data protection law, GDPR, in 2016. Graph and subgraph isomorphism are commonly used on privacy attacks that disclose personal information, like the relationship between people. For example, consider a network where the relationships are private and represent sensitive information, like email exchange (node are people and edges represent if they exchanged emails). This information should only be known by the people who are, in fact, exchanging the emails, and not by anyone else. However, attackers want to disclose this information.

Social network platforms sometimes release their network to allow for empirical studies. However, to protect their users' privacy, the graph is first anonymized, i.e., the personally identifiable information (PII) of every node is removed. A famous case occurred in 2009 with the Netflix Prize [12], where the company created an online competition to develop a recommendation system that improved over the one currently being used by Netflix and, so, they released an anonymized graph of users and movies. Although the network was anonymized before being released, the network structure is often enough to build a privacy attack and identify some (or all) of the nodes, hence identifying the relationship between them, which is usually sensitive and private information. One common privacy attack on anonymized networks

is based on graph isomorphism: suppose the attackers belong to the network and, hence, know the subgraph induced by the nodes they control. They can now run a subgraph isomorphism algorithm to find this subgraph on the anonymized graph and then start identifying other nodes using prior information, e.g, nodes connected to the attackers, the node with the highest degree, and so on [13–15].

Besides anonymizing the network, it is common for the network's holders to remove some edges and nodes from it before releasing it, making attacks harder. This problem can not be solved with isomorphism anymore since the subgraph known by the attackers might be different. However, it is still possible to solve this problem using graph matching techniques to search for the known subgraph and disclose other relationships as well.

1.1.1 Motivation

A limitation of graph isomorphism is that it determines if two graphs have the exact same structure. If a single edge is missing in a copy of a very large graph, then isomorphism fails. In real-world, graphs rarely have a perfect structure and, when collecting data to generate a graph, it is common to miss some edges, creating a graph that is not an exact copy of a previously collected copy, but that is very similar. So, how to determine if two graphs are similar?

The graph isomorphism problem belongs to a larger class of problem known as graph matching or network alignment [16, 17]. Graph matching is the problem of finding the best similarity between two graphs by matching the nodes of the two graphs as to minimize the number of edges not appearing on both graphs. Note that when this number is zero the two graphs are isomorphic.

Applications of graph matching problem are similar to the ones of the isomorphism problem since in many scenarios the problem becomes finding the best match between two different graphs. Once again, take social networks for example. The structure of two different social networks among the same set of people can be similar, since users that communicate through WhatsApp are likely to also communicate through Facebook, for example. However, it is very unlikely that these two networks are isomorphic.

The existing algorithms for graph matching simultaneously explore the structure of both graphs to align the nodes. For example, aligning the nodes that have the largest degrees off both graphs [18]. However, a different and not much-explored approach is to generate a canonical labeling for each graph independently in a way that the canonical labelings of structurally similar graphs are similar. However, the current existing canonical labeling algorithms do not consider this constraint as their main focus is solving the graph isomorphism problem. Current canonical

labeling algorithms are all very dependent on the graph structure and not robust to edge removal. Removing edges from a graph yields a very different canonical labeling if compared to the canonical labeling of the original graph. For example, the canonical labeling of a graph G_1 and the canonical labeling of a graph G_2 generated after removing a single edge from G_1 can be very different from each other. However, graphs have structural features that are robust to small perturbations on its structure (like edge removal) that could be used to drive canonical labeling algorithms. One such feature in many real networks is node distance.

Social networks or collaboration networks, for example, often have many different paths with the same length, in such a way that the shortest path between nodes will not change if some edges are removed from the graph. In other words, if one edge is removed and the shortest path between two arbitrary nodes used this edge, there is likely another path with the same distance that does not require the edge that was removed. This opens a new path to explore canonical labeling techniques that are based on distances, which is the central theme of this dissertation. The idea is to use distances between nodes to differentiate them and then relabel them based on that.

1.1.2 Contributions

This dissertation presents four different canonical labeling algorithms, two based on degree and two based on distances. The degree based are independent implementations of a classic algorithm and a variation developed within this research. Since degrees are not very robust to edge removal, the main focus of this work is on distance based canonical labeling algorithms with applications to the graph matching problem.

This work main contributions are:

- A variation of one state of the art degree based canonical labeling algorithm.
- Two novel distance based canonical labeling algorithms. To the best of our knowledge, these are the first canonical labeling algorithms fully based on node distances.
- Proof of correctness of the proposed algorithm when applied to the isomorphism problem.
- Proof of correctness for the graph matching problem under a given necessary condition.
- Implementation and evaluation of the algorithms on real and synthetic networks.

- Empirical comparison between these algorithms and four other algorithms, two based on degrees developed by this research and two state of the art canonical labeling algorithms, indicating the superiority of the proposed distance based approaches.

1.2 Organization

Chapter 2 covers the concepts that are used throughout this dissertation and some related work, including a brief overview of canonical labeling and graph matching problems. Chapter 3 presents two implementations, a classic canonical labeling algorithm and a proposed variation based on degrees. Chapter 4 details the distance based canonical labeling algorithm and presents the theoretical proofs. Chapter 5 presents the results of six different canonical labeling algorithms on different graphs and different scenarios of edge removal when considering the graph matching problem. Chapter 6 presents the conclusion and some future work.

Chapter 2

Basic Concepts and Related Work

2.1 Important Definitions

The following notation and definitions will be used throughout this work. Let $G = (V, E)$ denote a graph with node set V and edge set E, assumed to be undirected. Let $n = |V|$ and $m = |E|$ denote the number of nodes and edges of G, respectively. Moreover, let $N_u = \{v|(u, v) \in E\}$ denote the neighbors of $u \in V$, and let $d_u = |N_u|$ denote the degree of u. Let $d(u, v)$ denote the distance as in the length of the shortest path between nodes $u, v \in V$. Since G is undirected, $d(u, v) = d(v, u)$.

Let $\tau_G : V \to [1, n]$ denote a bijective function between the nodes of G and the respective range of natural numbers. Note that τ_G depends on G and is called the canonical labeling of G. Thus, $\tau_G(u)$ is the canonical labeling of node u in G. Let $G_\tau = (V_\tau, E_\tau)$ denote the graph permutated by the canonical labeling τ_G, with node set $V_\tau = \{\tau_G(u)|u \in V\}$ and edge set $E_\tau = \{(\tau_G(u), \tau_G(v))|(u, v) \in E\}$.

To summarize it all, Table 2.1 presents the notations and definitions given.

Notation	Definition
$G = (V, E)$	Graph with node set V and edge set E
n	Number of nodes of G
m	Number of edges of G
N_u	Neighbors of $u \in V$
d_u	Degree of u
$d(u, v)$	Distance between nodes $u, v \in V$
τ_G	Canonical labeling of G
$\tau_G(u)$	Canonical labeling of u
$G_\tau = (V_\tau, E_\tau)$	G permutated by τ_G

Table 2.1: Notations and definitions

2.1.1 Partitions

The following definitions give the most important concepts regarding partitions used in this work.

Definition 2.1.1. *A partition π of a set S is a set of disjoint non-empty subsets of S whose union is S.*

Definition 2.1.2. *An ordered partition is a sequence of subsets $[S_1, S_2, \ldots, S_r]$ such that the subsets form a partition of S but the order of the subsets in the sequence is relevant. Moreover, each subset S_i is considered to be a sequence such that $S_i = [S_{i1}, S_{i2}, \ldots, S_{ik_i}]$ and this order is relevant.*

To better illustrate, let $S = [1, 2, 3, 4, 5, 6]$. A possible ordered partition is $\pi_1 = [[1, 2], [5, 4, 6], [3]]$, such that $S_1 = [1, 2]$, $S_2 = [5, 4, 6]$ and $S_3 = [3]$. Note that $\pi_2 = [[1, 2], [3], [5, 4, 6]]$ and $\pi_3 = [[2, 1], [3, 4, 5], [6]]$ are two different ordered partitions.

The elements of a partition are usually called cells. A singleton cell is a cell with cardinality one, such as S_3 in the above example, and a discrete partition is a partition with only singleton cells, such as $S = [[2], [1], [5], [3], [6], [4]]$. For a set S with n elements, there are $2^{n-1}n!$ different ordered partitions, since there are 2^{n-1} possible ways of splitting the cells and $n!$ possible permutations of the elements.

Definition 2.1.3. *Let π_1 and π_2 be ordered partitions of S. π_1 is finer than π_2 if:*

- $V_i \subseteq W_k, \forall\, V_i \in \pi_1, W_k \in \pi_2$
- *For two cells $V_i, V_j \in \pi_1$ with $i \leq j$, and two cells $W_k, W_l \in \pi_2$ such that $V_i \subseteq W_k, V_j \subseteq W_l$, then $k \leq l$*

Definition 2.1.4. *If π_1 is finer than π_2, then π_2 is coarser than π_1.*

To exemplify the two definitions given above, consider $\pi_i = [[1, 2], [3, 4], [5, 6]]$ and $\pi_j = [[1, 2, 3, 4], [5, 6]]$. In this scenario, π_i is finer than π_j: π_i has three cells, $V_{i_1} = [1, 2], V_{i_2} = [3, 4]$ and $V_{i_3} = [3, 6]$ and π_j has two cells, $V_{j_1} = [1, 2, 3, 4]$ and $V_{j_2} = [5, 6]$. All cells from π_i are contained in π_j ($V_{i_1} \subseteq V_{j_1}$, $V_{i_2} \subseteq V_{j_1}$ and $V_{i_3} \subseteq V_{j_2}$) and the order of the cells is respected ($V_{i_1} \subseteq V_{j_1}$ and $V_{i_3} \subseteq V_{j_2}$, for example). Therefore, π_j is also coarser than π_i.

The partitions used in this dissertation are always over the set of nodes V of a graph, and thus the notation for representing the cells is V_i.

2.1.2 Equitable Partitions

Equitable partitions are partitions where the cells conform to a specific set of rules or conditions. These rules can differ for each type of algorithm, but the main idea

is always the same. To better illustrate the concept of an equitable partition, we explore the definition presented on McKay's canonical labeling algorithm [6], which is reviewed in much more detail later on in this chapter.

Definition 2.1.5. *Let $\pi = [V_1, V_2, \ldots, V_r]$ denote an ordered partition of V in a graph G. Let $d_G(v, V_i)$ denote a function that returns the number of nodes in V_i that are adjacent in G to v, in particular, $d_G(v, V_i) = |V_i \cap N_v|$. An ordered partition is equitable if:*

- $\forall\, V_i, V_j \in V, \forall\, u, v \in V_i,\ d_G(u, V_j) = d_G(v, V_j)$

In other words, a partition is equitable if, for any two cells (not necessarily distinct) from a partition, all nodes from one cell have the exact same number of neighbors on the other cell.

For example, consider the graph shown in the upper left of Figure 2.3. A possible partition is $\pi = [[1, 3, 7, 9, 8, 6, 4, 2, 5]]$, i.e., all nodes in a single cell. Since $V_1 = [1, 3, 7, 9, 8, 6, 4, 2, 5]$, containing all nodes from G, this partition would only be equitable if G was a regular graph. Since this is not the case (for example: node 1 has degree 2 (hence $d_G(1, V_1) = 2$) and node 5 has degree 4 (hence $d_G(5, V_1) = 4$)), π is not an equitable partition. A possible equitable ordered partition for this graph is $\pi' = [[1, 3, 7, 9], [2, 4, 6, 8], [5]]$.

Every possible partition π can be turned into an equitable partition π'. Later on in this chapter there is an explanation of an algorithm that takes a partition π and returns an equitable partition π' that is finer than π.

2.2 Graph Isomorphism Problem

Definition 2.2.1. *An isomorphism between two graphs $G = (V, E)$ and $G' = (V', E')$ is a bijection between the vertex sets of G and G' $f : V \longrightarrow V'$ such that $(u, v) \in E$ if and only if $(f(u), f(v)) \in E'$ [19, 20]. Note that f is a transformation of the node set that preserves all edges.*

An example of two isomorphic graphs and its bijection is shown in Figure 2.1.

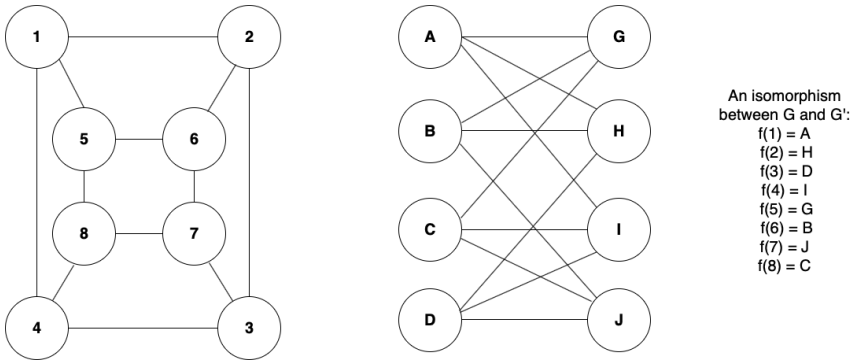


Figure 2.1: Example of an isomorphism between two graphs.

Finding an isomorphism between two graphs G and G' is a fundamental problem in theoretical computer science, and no polynomial time algorithm exists for the general case. However, there are trivial necessary conditions for the existence of an isomorphism. For example, it is necessary that $|V| = |V'|$, $|E| = |E'|$ and, moreover, the ordered degree sequence must be the same on both graphs. The ordered degree sequence is an ordered numerical sequence containing the value of the degree of all nodes. If this sequence is not the same on two graphs, they are not isomorphic since there is at least one node from one graph that is not correctly represented on the other graph. All the presented conditions are necessary, but not sufficient.

2.2.1 Automorphism

Automorphism is a problem derived from isomorphism. Consider the following definitions for the main concepts behind this problem.

Definition 2.2.2. *An automorphism is an isomorphic mapping from a graph to itself [21].*

An example of an automorphism is presented in Figure 2.2, where it is possible to see that nodes 2 and 3 can change their positions (creating a new graph) while preserving the edge set.

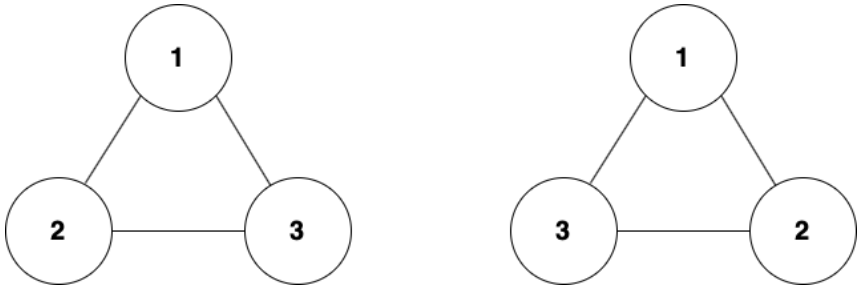


Figure 2.2: Example of an automorphism.

2.3 Graph Canonization and Canonical Labeling

Graph canonization is the problem of finding a canonical form to write G. A canonical form or canonical labeling of G is a labeled graph G_τ that is isomorphic to G and represents the entire isomorphism class of G. In other words, canonical labeling is a way of labeling the nodes such that isomorphic graphs are identical, i.e., have the same edge set. Canonical labeling algorithms can solve the isomorphism problem since it suffices to generate a canonical labeling for each of two graphs G_1 and G_2 and then check if the new labeled graphs G_{τ_1} and G_{τ_2} are the same, i.e., have the same edge set (which can be done in $\mathcal{O}(n^2)$, for a graph with n nodes).

Another common definition for a canonical labeling function is to provide a way of sorting the nodes of a graph G. For instance, G_τ is a n sized permutation, chosen by some criteria that depend on the structure of G from all possible $n!$ permutations.

In Figure 2.3, the two graphs in the upper row are clearly (visually speaking) isomorphic, though they are not identical since their edge set is different (for example, edge (1,2) in the left graph is not in the right graph). The canonization of the graphs in the upper row of Figure 2.3 are shown in the lower row of Figure 2.3, which are identical (have the same edge set).

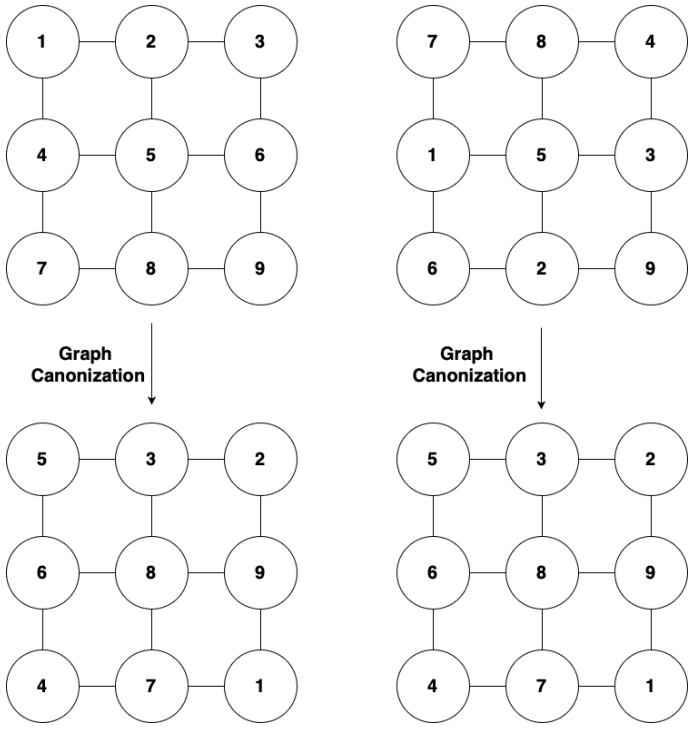


Figure 2.3: Canonical labeling example with two graphs that are 3x3 grids.

2.3.1 McKay's Algorithm

There are many algorithms to generate a canonical labeling for a graph, and one of the most famous was designed by McKay, first described in 1980 [6]. Besides

describing this algorithm, the next subsections will also describe other solutions to the problem.

McKay's algorithm generates a canonical labeling for a graph G. The main strand of this algorithm is to determine an ordering for the nodes based on the node's degree and neighbors. This algorithm influenced many subsequent approaches, including the one proposed in this work, so a detailed explanation is given below.

McKay's algorithm receives a graph G as input and generates an initial single-cell partition V, with all n nodes of G. Suppose G is the graph shown in the upper left Figure 2.3, so $n = 9$ and $V = \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Then, the initial partition is $\pi = [[1, 2, 3, 4, 5, 6, 7, 8, 9]]$, where nodes are taken in order. The purpose of the algorithm is to work on π until it reaches an ordered discrete partition π' with n singleton cells (this procedure will soon be explained). This ordered discrete partition is then a candidate for the canonical labeling of the graph, τ_G. In particular, each cell in π', or τ_G, represents a new label for a node in G, according to its position in the ordered discrete partition. Following the example, the resulting canonical labeling is $\tau_G = [[1], [3], [7], [9], [8], [6], [4], [2], [5]]$, such that node 1 is relabeled to 1, node 2 is relabeled to 8, node 3 is relabelled to 2 and so on. Thus, each node v is relabelled according to its index in the canonical labeling generated by the algorithm.

Refinement Procedure

A refinement procedure, which was first described by McKay in 1976 [5], is a function that receives an ordered partition and transforms it on an equitable ordered partition. This is done by breaking cells that do not have all nodes with the same degree towards another cell, and such nodes become new cells. For a cell V_k and a cell W, it will split V_k into new cells $(X_1, X_2, \ldots, X_s)$ such that for any pair of nodes $x \in X_i$ and $y \in X_j$, $d_G(x, W) < d(y, W)$ if and only if $i < j$. This procedure of generating new cells from an existing cell is called shattering. If $s = 1$, then all nodes from V_k have the same number of neighbors in cell W and there is no need to split V_k at this point.

The refinement procedure presented in [5] is next described. It receives a partition π, that is transformed into an equitable partition. The procedure verifies all ordered pairs of cells as described above and splits the cells accordingly, creating new cells and modifying the partition. It will use α as an assistant set, initialized with all cells in π, used to select the pair of cells. The algorithm will take a cell W from α, a cell V_k from π and check if V_k is shattered by W. If this is not the case, the algorithm moves to the next cell from π. If V_k is the last cell from π, it moves W to the next cell in α and continues from the first cell in π, V_1. However, if a shattering occurs, then π needs to be updated, removing V_k and adding the ordered

split cells generated by V_k in its place. Moreover, all new cells generated are added to the end of α, as they now belong to the current partition and will be needed to check if they can shatter any cell from π. The process of appending the new cells to α has just one more detail: if V_k is in α in a position after W that has not been considered yet (remembering that the algorithm works in the cells in α in order, so this means that V_k is a position greater than the one currently being considered) this cell is removed from α, since it is no longer a cell in π.

Algorithm 1 Refinement Procedure

Input: ordered partition π, graph G
Output: equitable ordered partition π'
$\pi' \leftarrow \pi$
$\alpha \leftarrow \pi$
$M \leftarrow |\alpha|$
$m \leftarrow 1$
while π' is not discrete and $m \leq M$ **do**
    $W \leftarrow alpha[m]$
    $m \leftarrow m + 1$
    $k \leftarrow 1$
    $r \leftarrow |\pi'|$
    while $k \leq r$ **do**
        $V_k \leftarrow \pi'[k]$
        Split V_k in $(X_1, X_2, \ldots, X_s)$ such that for any $x \in X_i$ and $y \in X_j$, $d_G(x, W) < d_G(y, W)$ if and only if $i < j$
        if $s \neq 1$ **then**
            $t \leftarrow \min\{j | \, |X_j| \geq |X_l|, \forall l \in [1, s]\}$
            for $i \leftarrow 1, i \leq s, i++$ **do**
                if $i = t$ **then**
                    if $alpha[j] = V_k$ for some $j (m \leq j \leq M)$ **then**
                        $alpha[j] \leftarrow X_t$
                    else
                        $\alpha.Append([X_i])$
                else
                    $\alpha.Append([X_i])$
            $M \leftarrow M + s - 1$
            Update π' by replacing V_k for $X_1, X_2, \ldots, X_s$ in that order
        $k \leftarrow k + 1$

This procedure is executed for all pairs of cells (one from α, another from π), up-

dating the partition with the new cells that are shattered. Once the algorithm checks all pairs of cells, the resulting partition is equitable and is returned. Algorithm 1 shows the pseudo-code of the refinement procedure.

Search Tree

To find the canonical labeling for a graph, McKay's algorithm builds a search tree where each node is an equitable ordered partition. The root of this tree is the result of the refinement procedure for the initial partition (the one with a single cell containing all nodes). The algorithm will start the exploration on the root node and will run a depth-first search on the tree to reveal other nodes. Each node representing a partition π will have children nodes, unless π is a discrete ordered partition. The children of a node are ordered partitions formed by removing each element e from the first non-trivial cell of π of the smallest size and placing e in a new cell. If the smallest non-trivial size cell in π has size t, the node representing π will have t children in the tree. For example, consider π as $[[1,2,3],[4,5,6,7],[8,9,10]]$. Then, the first non-trivial cell of the smallest size is $W = [1,2,3]$. The children of π are $[[1],[2,3],[4,5,6,7],[8,9,10]]$, $[[2],[1,3],[4,5,6,7],[8,9,10]]$ and $[[3],[1,2],[4,5,6,7],[8,9,10]]$. Each child partition becomes the input to the refinement procedure generating the corresponding equitable partition as the node on the search tree. A part of the search tree for this graph is presented in Figure 2.4.

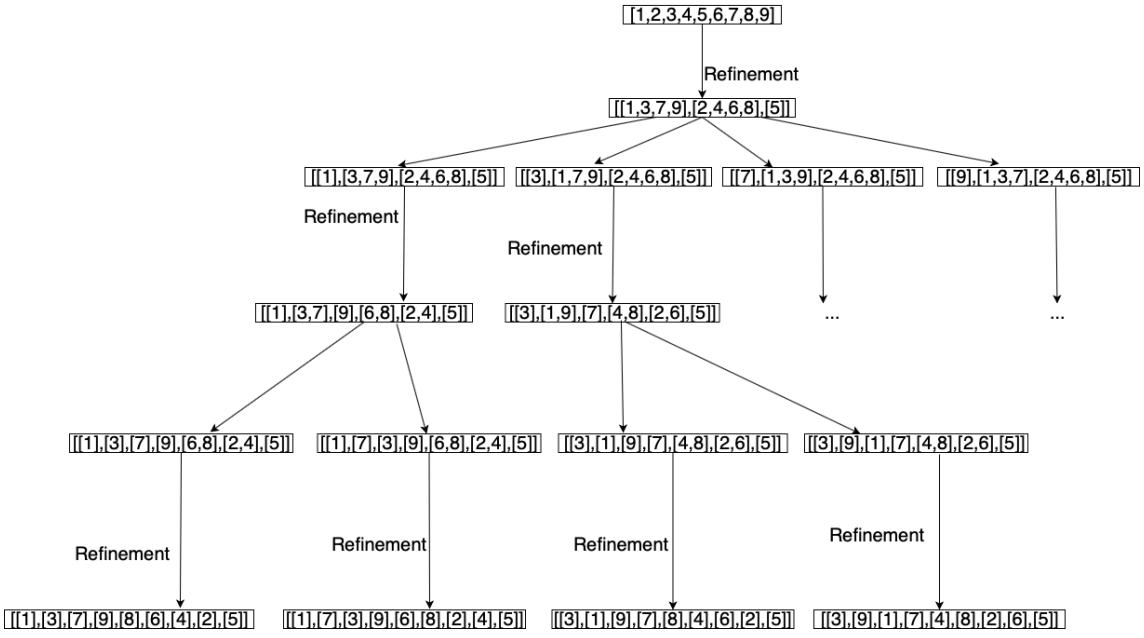


Figure 2.4: Part of the search tree for the 3x3 grid graph.

Every time a discrete partition is found (leaf node), a binary representation for this canonical labeling is generated. The discrete partition π represents a graph G_π

isomorphic to G and the binary representation of G_π is a binary number generated by the superior triangle of the adjacency matrix of G_π. Since the algorithm must return a single canonical labeling, the idea is to choose amongst all the possible binary representations, such as returning the largest binary representation, since it can be easily compared with other representations. The algorithm keeps track of all different binary representations found.

Note that two different discrete partitions and their corresponding canonical labelings can have the same binary representation. This means that both canonical labelings represent a graph with the same adjacency matrix and, hence, are identical, which means the graph has automorphisms. Thus, it does not matter which canonical labeling is returned by the algorithm in this case.

Pruning

There are many reasons for the search tree to grow and become very large. For example, if the input graph G has a large number of automorphisms since the number of leaves in the search tree is at least the number of automorphisms. However, although the canonical labeling will be different between different automorphisms, the graphs they represent are exactly the same and, hence, so are the corresponding binary representations. Since the search tree is explored using a depth-first procedure, the algorithm can prune the tree based on the automorphisms that it finds, avoiding the generation of unnecessary paths that lead to leaves representing the same graph (same binary representation). The pruning method only removes sections of the tree that would generate a canonical labeling that has already been seen or that are smaller than the best canonical labeling found so far (for example, if the algorithm wants to return the canonical labeling associated with the largest binary representation and that path will only generate canonical labelings with binary representations smaller than the largest one yet found).

When exploring a new leaf t, the algorithm generates the binary representation of the corresponding canonical labeling. If this binary representation is identical to any of the binary representations already found, then the search tree is pruned. In this case, there is a leaf p already discovered with this same binary representation. Let a be the node in the tree that is the deepest common ancestor of p and t, and let b be the child of a that is ancestor of p and c be the child of a that is ancestor of t, as shown in Figure 2.5. Node p has a discrete partition π_p associated to it, and node t has a discrete partition π_t. Since π_p and π_t represent the same graph, there is a permutation γ between p and t. The permutation γ that sends p to t also sends b to c. In other words, the subtree of the search tree rooted at b is isomorphic to the subtree rooted at c, thus, the leaf nodes, or discrete partitions, that have c as an ancestor are the same as the leaf nodes that have b as an ancestor (under

permutation γ). Since this search tree is created depth-first, the tree rooted at b is examined before the tree rooted at c and once this permutation between b and c is found, there is no more reason to continue examining the subtree rooted at c, so the algorithm prunes node c and all its children and continues the search from node a.

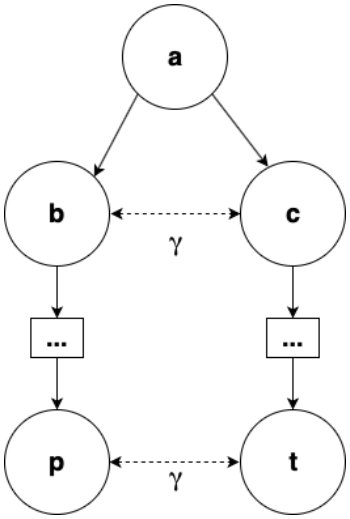


Figure 2.5: Pruning example.

Another way to prune the search tree is based on the magnitude of the prefix of the binary representation, called prefix-based pruning. Before exploring a node of the tree that is not a leaf, the algorithm generates the prefix of its binary representation using the partition associated with the node. To illustrate how this prefix is generated, consider the graph in Figure 2.6 and a partition $\pi = [[1], [2], [3], [4], [5, 6, 7], [8, 9]]$.

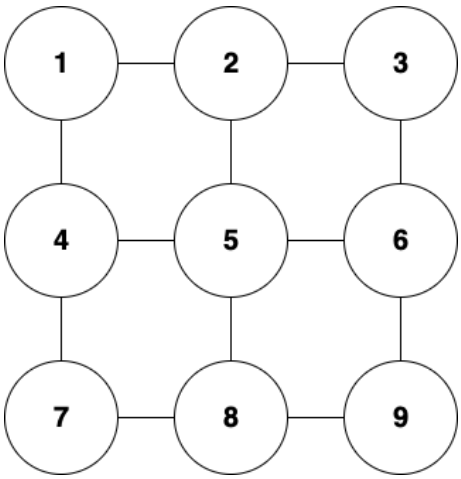


Figure 2.6: 3x3 grid graph example.

Since the partition π has singleton cells, the algorithm already knows the label of some of the nodes as determined by the canonical labeling represented in this partition, and also on all children nodes (since all of them have the same singleton

cells in the same order). Therefore, the binary representation of the prefix can be generated using the singleton cells and construct the initial bits of the adjacency matrix. To better illustrate, consider the graph in Figure 2.7, that represents the partial canonical labeling graph represented by the partition π given above. Note that the algorithm already knows that nodes 1 and 2 are neighbors, nodes 1 and 3 are not and that nodes 1 and 4 are neighbors. So, the prefix of the binary representation associated with this partition is 101. All leaf nodes that are descendants of this node in the search tree will also have this prefix in their respective binary representations, based on the way the prefix is generated and the fact that the cells do not change their order on the partition.

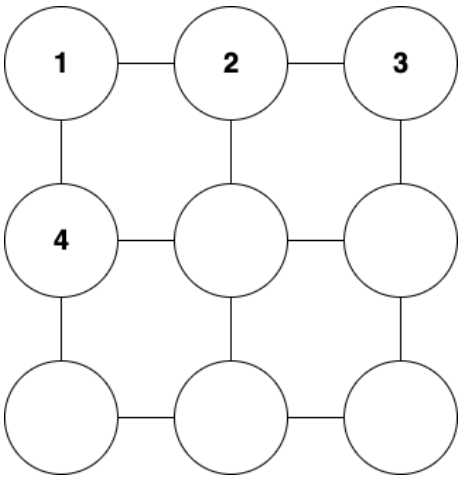


Figure 2.7: Partial canonical labeling graph based on a not discrete partition.

Recall that the algorithm returns the canonical labeling associated with the largest binary representation. Thus, if the algorithm finds a node with a binary representation that has a prefix that is smaller than the largest binary representation yet found (considering only the first bits of this representation, making it the same size as the prefix), this node does not need to be explored anymore and it can be pruned from the search tree. All future children of this node will share the same prefix of its binary representation, so they will all have a smaller representation than the largest one and, hence, do not need to be considered.

Besides these two pruning techniques, there are other more complex and effective procedures, decreasing the number of nodes on the search tree and, hence, improving the efficiency of the algorithm. Chapters 3 and 4 cover our degree and distance based canonical labeling algorithms, where other pruning techniques have been developed.

Complexity

Although the graph isomorphism problem is clearly in NP (since an explicit isomorphism serves as a certificate that two graphs are isomorphic and this can be

done in polynomial time [22]), it is still not known if this problem is either solvable in polynomial time or NP-Complete. Thus, the described algorithm to generate a canonical labeling and find if two graphs are isomorphic has exponential running time on some inputs (worst-case scenario) [23], but in general, it performs well (for example, graphs of bounded degree can be computed in polynomial time [4]). Hence, the algorithm's complexity will depend on the graphs' classes.

2.3.2 Nauty and Traces

Nauty and Traces [7] are classic programs for generating canonical labelings. They are both widely used, written in C [24] and considered very efficient implementations. This work uses nauty to generate canonical labelings, as it will be shown in Chapter 5. Both programs are derived from McKay's canonical labeling original algorithm, described in Section 2.3.1 and received many improvements over the years [5, 25]. Traces is actually very similar to nauty and its innovative approach outperforms other programs and implementations on difficult graph classes [7].

2.3.3 Bliss

Bliss in an open-source tool for computing canonical labelings of graphs, and its algorithms and data structures were published in 2007 [8]. Its implementation in Python, which allows fast prototyping of algorithms for isomorphism rejection, for instance, is used in this dissertation, as discussed in Chapter 5. Bliss is built especially for fast handling of large and sparse graphs, running a backtracking algorithm based on individualization, refinement, efficient data structures and subroutines, and pruning heuristics.

The idea of the algorithm is similar to nauty and the main contribution of Bliss is a design that can accommodate large graphs and facilitate fast indexing on the search tree. In many cases, Bliss can avoid a linear time overhead when processing a node in the search tree by additional bookkeeping and by accessing only essential parts of the ordered partition and the graph. It also improves the heuristics used to prune the tree, facilitating early pruning by means of an incremental leaf certificate [8].

2.4 Refinement Strategies

There are several strategies to verify graph isomorphisms and many use refinement strategies, that are briefly covered in this section. However, to the best of our knowledge, such strategies have the goal of improving the efficiency of the algorithm either by reducing the complexity, decreasing the number of nodes in the search

tree, making the refinement stronger to split more cells, etc. Refinement strategies have not been proposed with the goal of identifying structure similarities in two or more graphs, beyond isomorphism.

2.4.1 Degree Sequence

This is one of the most successful and well-known techniques used in most practical graph isomorphism algorithms [23, 26, 27] and terminates in polynomial time for almost all graphs [9]. The idea and examples are discussed below.

Given a graph $G = (V, E)$ with n nodes, the idea is to give each node $v \in V$ a label $l(v)$, such that each node gets a unique label. The first step is to label each node with its degree. The algorithm is iterative, generating a sequence of labels for every node. In the next iteration, the new label $l'(v)$ of node v is defined by the labels of its neighbors:

$$l'(v) = \{l(v), \{l(u)|(u, v) \in E\}\} \qquad \forall v \in V$$

The Weisfeiler-Lehman [28] algorithm uses this procedure to check if two graphs G and G', with n nodes and m edges, are isomorphic. The idea is to process both graphs simultaneously until the node label sets of G and G' differ, which means the graphs are not isomorphic. Moreover, if the number of iterations reaches n, then the graphs are either isomorphic or the algorithm was not able to determine that they are not isomorphic. The complexity of the Weisfeiler-Lehman algorithm with h iterations is $\mathcal{O}(hm)$ and, in the worst-case scenario, $\mathcal{O}(mn)$.

Although this algorithm works in polynomial time and the isomorphism problem is known for not having a polynomial-time algorithm, it, unfortunately, does not work for all graphs. There is a quite obvious class where this algorithm fails: regular graphs, where all nodes have the exact same degree. Other graphs that cannot be distinguished by this algorithm are covered by Cai et al. [29].

2.4.2 Distance Sequence

For random graphs, the degree refinement presented above is not a good strategy to distinguish the nodes, as many nodes have the same (randomly applied) degree. Thus, another attribute is often used, the distance sequence [30]. For a node v, let $d_i(v)$ denote the number of nodes at distance i from v. The distance sequence of node v is the sequence of $d_i(v)$ for $i = 0, 1, 2, \ldots, n$. The refinement (iterative procedure) itself is the same as presented for degree sequence.

2.5 Graph Matching Problem

There are two completely different graph matching problems in network science, so the first step is defining them both.

In graph theory, a matching M in a graph $G = (V, E)$ is a set of pairwise non-adjacent edges. This definition is applied to several problems and one of the most famous is finding the maximum matching of a graph, which is a matching that contains the largest possible number of edges. In an unweighted bipartite graph, the problem is solved in $\mathcal{O}(\sqrt{V}E)$ time [31].

The problem we are interested in, however, is the problem of finding a structural similarity between two graphs. The idea is to determine a correspondence between the node set of the two graphs such that the edge sets are preserved as much as possible. This problem generalized graph isomorphism as it does not require the graph to be isomorphic to find a correspondence. Just as clarification, this is the problem this dissertation will be referring to as graph matching.

There are many algorithms to solve the graph matching problem but none are based on canonical labeling and all receive as input the two graphs that must be matched. An approach that is more similar to canonical labeling is known as Reconciling Graphs and will be discussed in the following. The main idea to reconcile the graphs is to define a signature for each node for both graphs being compared and then the signatures between the graphs are compared.

This section will also discuss how graph matching is being used on other applications, like creating privacy attacks on social networks.

2.5.1 Reconciling Graphs

The idea of reconciling graphs is to change the edge set of one graph such that it becomes isomorphic to another. In particular, the problem considers two unlabeled graphs $G_A = (V_A, E_A)$ and $G_B = (V_B, E_B)$, where $|V_A| = |V_B| = n$ and only $d \ll n$ (also, $d \ll m_a$ and $d \ll m_b$) edges are changed (added or deleted) in E_A to make G_A isomorphic to G_B. That is, there are two different but similar graphs, with a small number of edges d that are different.

One recent approach [18] for random graph reconciliation, specifically for Erdõs-Rényi random graphs [32], finds a signature scheme for the vertices where each node's signature is invariant under relabeling and with high probability every vertex in a graph has a unique signature. This approach is different from the canonical labeling approach for one main reason: two or more nodes can have the same signature (depending on the graph), while with canonical labeling each node has a unique label. However, the idea is the same: if two graphs have different signature sets (the edge set of the relabelled graph is different) they are not isomorphic. For random

graph reconciliation, the signature scheme has an additional property that makes the signatures robust to changes to a small number of edges in the two graphs.

The first step of the algorithm is ordering the vertices by degree so that $d(v_1) \geq d(v_2) \cdots \geq d(v_n)$. For the h nodes with the largest degree (h is a parameter), their signature becomes their degree ordering (the node with the largest degree is labeled 1, the node with the second largest degree is labeled 2 and so on). For the remaining $n - h$ nodes, each node u receives a signature $sig(u)$ that is an h-bit string where the ith bit of the string denotes whether or not node u is adjacent to v_i, the node with the ith larger degree.

Consider the following definition for a class of graphs.

Definition 2.5.1. *A graph is (h, a, b)-separated if, after sorting the nodes by their degrees ($d(v_1) \geq d(v_2) \geq \cdots \geq d(v_n)$):*

- $d(v_i) - d(v_{i+1}) \geq a, \forall i \in [1, h-1]$
- *For all $i, j \in \{h+1, n\}$ and $i \neq j$, the Hamming distance between $sig(v_i)$ and $sig(v_j)$, $|sig(v_i) - sig(v_j)|$ is at least b. The signature $sig(v_i)$ is a binary data string and the Hamming distance is a way of comparing two binary strings with the same size by counting the number of bits positions in which the two bits are different.*

The article shows that for an appropriate set of parameters, a random graph is $(h, d+1, 2d+1)$-separated with high probability. Let G denote an Erdõs-Rényi random graph generated and G_A and G_B denote graphs obtained by making at most $\frac{d}{2}$ edge changes to G. Let $w_1, \ldots, w_n$ be the nodes of G. If a node $v_a \in G_A$ and a node $v_b \in G_B$ correspond to the same node $w_i \in G$, then they conform. Then, if it is possible to label G_A and G_B in a way that every node from G_A that is labeled the same in G_B conform, there is a labeled graph reconciliation problem with at most d edges differences.

By definition, since G is $(h, d+1, 2d+1)$-separated and G_A and G_B differ by at most d edges, the h largest degree nodes in G_A conform to those of G_B, in the same degree order. For the other $n - h$ nodes, they use the set of sets reconciliation solution to reconcile the signatures and, then, reconcile the graphs.

2.5.2 De-anonymization of Social Networks

With the growing discussion about data privacy, social networks are continuously studying ways to protect their users' personally identifiable information. Typically, whenever a network is made available for empirical studies, for example, it first is protected by anonymization. But is that enough?

Recent works demonstrate that anonymizing the network might not be sufficient to protect the users' privacy. Some of these works discuss how the availability of node and link data from another network, that is correlated with the anonymized network available, can be used to re-identify the anonymized nodes with graph matching techniques [33–35]. The idea is that the attackers might have an auxiliary labeled network, in which user identities are known (taken from a public source, for example), that is correlated to the anonymized network released, i.e., there are a number of users on both networks and their behavior is similar on them. With the two networks, then it is possible to apply graph matching algorithms to find the best match between them and re-identify some, or all, nodes from the anonymized network.

One more theoretical work aims to answer the question of whether de-anonymization is impossible in an inherently anonymous and sufficiently sparse network, with unlimited computational power and access to an auxiliary labeled network [34]. Here, there are no worries about the computational complexity of the process, just about the feasibility and limits for de-anonymization regardless of the algorithm employed.

The idea is to introduce an Erdõs-Rényi random graph $G(n, p)$ with an extra parameter s, that controls the correlation between the two networks over the same node set. For a fixed realization of $G(n, p)$, two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ are generated. Each edge $e \in E$ is in the edge set of E_1 and E_2 with probability s, which makes G_1 and G_2 also Erdõs-Rényi random graphs, but with correlated edge sets. Now, the goal is to determine if, with only the anonymized version of G_1 and G_2 at disposal, is possible to find the correct mapping between the nodes of the two graphs. The graph matching problem is then defined as follows: let π denote a permutation on the node set V, which has $n!$ ways. The identity permutation, denoted by π_0, is the correct mapping between the node set of G_1 and G_2. Now, it is only left to define an error function Δ_π over the set of all permutations, which succeeds if it is uniquely minimized by π_0:

$$\Delta_\pi = \sum_{e \in E_1} \mathbb{I}(\pi(e) \notin E_2) + \sum_{e \in E_2} \mathbb{I}(\pi(e) \notin E_1)$$

With this error function, they prove that if the sampling probability s is beyond some threshold, as the number of nodes n grows large, the identity permutation π_0 indeed minimizes the error function, which is great theoretical results for large networks.

On the other hand, a more practical work by Narayanan and Shmatikov [33] developed a de-anonymization algorithm based purely on the network topology that uses the structural similarity between the anonymized network and an auxiliary

network. To demonstrate its effectiveness on real networks, they show that a third of the users who had accounts on both Twitter and Flickr, two popular social networks, can be re-identified in the anonymous Twitter graph with only a 12% error rate, even though the overlap between the edge sets is less than 15%.

In their work, a social network S consists of a directed graph $G = (V, E)$. Let $S = (V, E)$ denote the anonymized released network and $S_{aux} = (V_{aux}, E_{aux})$ the auxiliary network, that the attackers had access to and whose membership partially overlaps with S. For this attack to work, the attackers also need to have detailed information about a very small number of nodes of S, which are called seed nodes. These nodes need to be present on both networks.

Since the aim of the algorithm is node re-identification, they define ground truth to be a mapping μ_G between the nodes V_{aux} and V. Node re-identification then simply refers to finding a mapping μ that succeeds on a node $v_{aux} \in V_{aux}$ if and only if $\mu(v_{aux}) = \mu_G(v_{aux})$. To measure the success of the attack, they assign a weight to each affected node in proportion to its node centrality in the network and take the fraction of the sum of weights correctly identified over the sum of all weights, capturing the effectiveness on the active nodes of the graph (which is the focus of the attack).

The first step of the algorithm is to identify the seed nodes on the target network, mapping them to each other. Then, a self-reinforcing propagation stage process takes place, in which the initial mapping is extended to new nodes using only the topology of the network, feeding the new mapping back to the algorithm. In summary, each iteration of the propagation algorithm starts with the accumulated list of mapped pairs between V and V_{aux}, then pick an arbitrary unmapped node $u \in V$ and computes a score for each unmapped node $v \in V_{aux}$, equal to the number of neighbors of u that have been mapped to neighbors of v. If the strength of the match is above a defined threshold, the mapping between u and v is added to the list and the next iteration starts. The result at the end is a large mapping μ between the two networks, which potentially re-identifies all mapped nodes.

2.5.3 Graph-Based Watermarking

Watermarks are added to software to define its owner and prove its authenticity, where the idea is to insert some data W (the watermark) into a program P so that in the resulting program it is not easy to detect and remove the watermark. Moreover, graph-based watermarking [36–38] is a technique where the data being inserted is a small graph, so graph algorithms can be applied on the software to recover the watermark and prove its authenticity, for example.

However, attacks are constantly made on these software, trying to prove that

the owner is not who the person claims to be. The attackers do not know where the watermark graph W is since it is a small hidden graph in the software, but they will run several pattern algorithms on it trying to discover it. They take a subgraph X assuming to be W, and slightly modify it, like removing a small number of edges (they do not want to modify the graph a lot because if they did not correctly found W they might impact the performance of the software). Since the graph has now been modified, it is not possible to find the watermark created and graph matching techniques on similar graphs (where one was generated after slightly modifying the other) can be used to retrieve the original watermark graph.

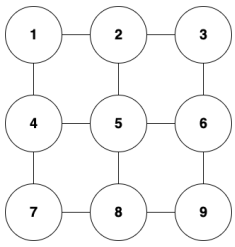
Chapter 3

Degree Based Canonical Labeling

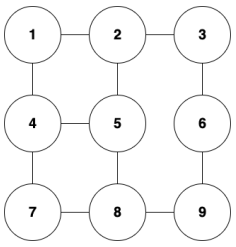
McKay's algorithm detailed in the previous chapter generates a canonical labeling for any input graph G that represents the entire isomorphism class of G, using only the neighborhoods of the nodes. More specifically, neighbors that a given node has in a given cell. This ensures that two isomorphic graphs will be the same graph after their respective canonization. But what if the graphs are not isomorphic?

Consider two graphs that are isomorphic except for a small number of edges (i.e., copy the graph and remove a small number of edges from it). Can McKay's algorithm tell that the graphs are similar? Are the resulting canonical labeling for the two graphs similar? Are the edge sets produced by both these canonical labelings similar?

As will be shown in Chapter 5, the answer to these questions, in general, is no. The resulting canonical labeling for similar graphs usually have very different edge sets. This happens because the only structural information used by McKay is the neighborhoods and, hence, nodes degrees, which is sensitive to edge removal. To illustrate this issue, consider the graphs shown in Figure 3.1. The graph in Figure 3.1b is identical to the graph in Figure 3.1a unless for the edge $(5, 6)$.



(a) A simple 3x3 grid.



(b) 3x3 grid with a missing edge.

Figure 3.1: Two identical graphs unless for a single edge.

The canonical labeling of both graphs as given by McKay's algorithm is shown below in Figure 3.2. There are 4 edges that are on both canonized graphs, only a 34% overlap between the two edge sets. This low percentage may lead to the

conclusion that the two graphs are structurally quite different, while in this case they are very similar.

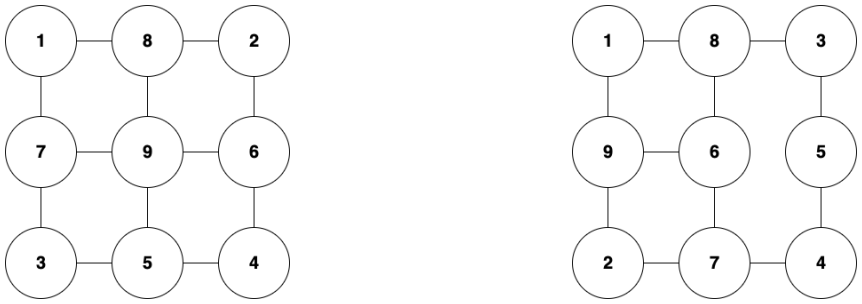


Figure 3.2: Canonical labeling of the graphs in Figure 3.1.

To understand the problem, consider the first partition from both graphs: $[[1, 3, 7, 9], [2, 4, 6, 8], [5]]$ for graph in Figure 3.1a and $[[1, 3, 6, 7, 9], [2, 4, 5, 8]]$ for graph in Figure 3.1b, where every node in each cell has the same degree. Those partitions are the result of the first interaction of the refinement procedure and they are already quite different from one another. Since one edge being removed reduces the degree of two nodes by one, their initial cells can change and introduce ambiguities. For example, the graph in Figure 3.1a has only one node degree 4, node 5. After the edge removal, its degree is 3 and it now shares a cell with three other nodes.

The problem is that the shattering procedure does not consider the graphs to be similar, so it splits the cells using the exact degree of the nodes. This can have drastic consequences in the search tree since the result of the first shattering will be different as well as all further nodes of the search tree.

3.1 Degree Similarity

The first idea to tackle the problem of identifying two similar graphs using canonical labelings is to change the shattering procedure. Instead of breaking cells by separating nodes with different degrees towards another cell, nodes with similar degrees could be grouped together. We called this approach degree similarity.

Definition 3.1.1. *Let δ_d be a positive integer named degree similarity. Two nodes u and v are said to have similar degrees if $|d_u - d_v| \leq \delta_d$.*

The refinement function is basically the same as the one described in Chapter 2, with the only difference being in the shattering procedure, which splits the cells to make the partition equitable. Consider the following definition for equitable partition in the degree similarity approach.

Definition 3.1.2. *A partition π is equitable in the degree similarity approach if:*

$$\forall\, V_i, V_j \in \pi$$

$$\forall\, u, v \in V_i$$

$$|d_G(u, V_j) - d_G(v, V_j)| \leq \delta_d$$

To better explain this idea, consider the graph in Figure 3.3 and $\delta_d = 1$. The initial partition is $[[1, 2, 3, 4, 5, 6]]$, with all nodes inside a single cell. The nodes degrees are $d_1 = 2$, $d_2 = 3$, $d_3 = 5$, $d_4 = 2$, $d_5 = 2$ and $d_6 = 2$, and the first shattering will group nodes $1, 2, 4, 5, 6$, since their degrees are classified as similar, and leave node 3 alone, generating the partition $[[1, 2, 4, 5, 6], [3]]$. The exploration of the search tree follows the same way as before.

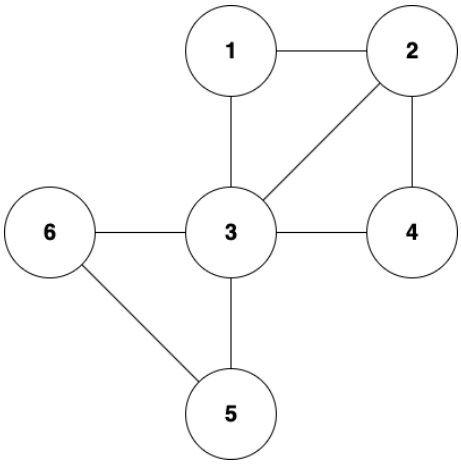


Figure 3.3: Example graph for the degree similarity procedure.

Now, lets remove the edge $(2, 3)$ from the graph in Figure 3.3, generating the graph in Figure 3.4. The nodes degrees are now $d_1 = 2$, $d_2 = 2$, $d_3 = 4$, $d_4 = 2$, $d_5 = 2$ and $d_6 = 2$, but the partition after the first shattering is $[[1, 2, 4, 5, 6], [3]]$, identical to the previous graph.

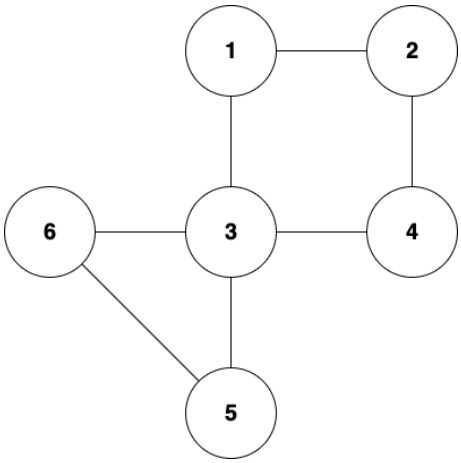


Figure 3.4: Example graph for the degree similarity procedure with one missing edge.

If we run the algorithm until the end, both executions (in each graph) will find the same canonical labeling represented by the discrete partition $[[2], [1], [4], [5], [6], [3]]$. Thus, this procedure correctly identified the similarity in the two graphs since their edge set after canonization is identical except for the missing edge. The main idea of this approach is for the shattering procedure to yield identical cells even when the two graphs are similar.

However, this approach also has drawbacks. First, the parameter δ_d must be defined and the larger the value, the larger is the search tree, since it becomes more difficult to split cells, and thus more nodes in the search tree will have more children. A larger search tree means a longer execution time of the algorithm. Even a $\delta_d = 1$ can already make the search tree much larger. Consider the graph in Figure 3.1a, where the degrees are 2, 3 and 4. The algorithm is unable to split the initial partition since $\delta_d = 1$, leaving no choice but to generate all possible children from this initial partition. This also happens in other search tree nodes for this graph, making the search tree huge. The second drawback is that this idea works only for some graphs and some edge removals. Depending on what is removed, the shattering for both graphs can become completely different. Consider the graph in Figure 3.5 with 10 nodes, eight with degree 1, one with degree 4 and one node with degree 6. The first shattering will create three cells: one for the node with degree 6, one for the node with degree 4 and one with all the eight nodes that have only one neighbor $[1, 2, 3, 4, 6, 7, 8, 9], [5], [10]]$. If any edge from node 10 is removed, for instance, $(1, 10)$, there will be 7 nodes with degree 1, one node with degree 0, one node with degree 5, and one node with degree 4. So, assuming $\delta_d = 1$, the result of the first shattering is $[[1, 2, 3, 4, 6, 7, 8, 9], [5, 10]]$. The algorithm grouped nodes 5 and 10 on the graph with the missing edge, which does not occur in the original graph. This kind of error in the initial partition, that also happens on other posterior

partitions, will lead to different search trees even when the graphs are structurally similar, and, hence, finding different canonical labeling.

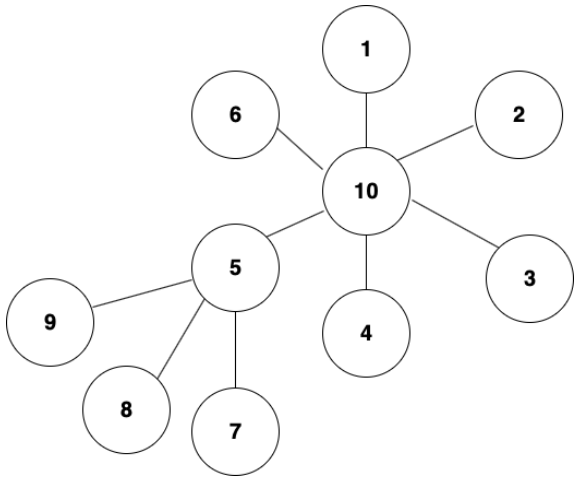


Figure 3.5: Example where the degree similarity procedure can fail.

To reduce the size of the search tree and reduce the running time, another parameter is now introduced. The idea is that nodes with a larger degree are more likely to lose an edge if compared to low degree nodes. This observation is true if edges are removed independently at random. Thus, larger degree nodes are more affected by edge removal, hence, having their degree changed to a smaller value. With this in mind, a new parameter p, $0 \leq p \leq 1$, is introduced and used only on the first shatter, when there is only one cell with all nodes. The algorithm considers the ordered degree sequence of the graph (ascending). Let d' denote the degree at the position $p * n$ in the ordered sequence. Now, for the first shatter, the algorithm only groups nodes with different degrees, using the δ_d, if their degrees are larger than d'. Nodes with a degree smaller than d' are only grouped if they have the same degree. If $p = 0$, then all degrees are grouped based on similarity. If $p = 1$ then no node is grouped based on similarity. Note that small values of p will generate a larger search tree (since splitting cells becomes harder). At the same time, increasing p will make the algorithm more similar to McKay's algorithm, where nodes are grouped only if they have the same degree. This parameter is helpful when the edges removed from the graph are the ones with the largest degrees and low degree nodes are not affected. In this case, low degree nodes will be on the same cells on the first partition on both graphs.

For example, consider the graph in Figure 3.3, with $\delta_d = 1$ and $p = 0.9$. The ordered degree sequence is $[2, 2, 2, 2, 3, 5]$. As mentioned before, the result of the first shatter when $p = 0$ will group the nodes with degrees 2 and 3 and leave the node with degree 5 alone. With $p = 0.9$, we have $d' = 3$ and, so, it does not group nodes with degrees 2 and 3, creating a partition with three cells. However, if $p = 0.2$, we

then have $d' = 2$ and, once again, nodes with degrees 2 and 3 are grouped together. This approach is really useful for larger graphs.

3.2 Implementation

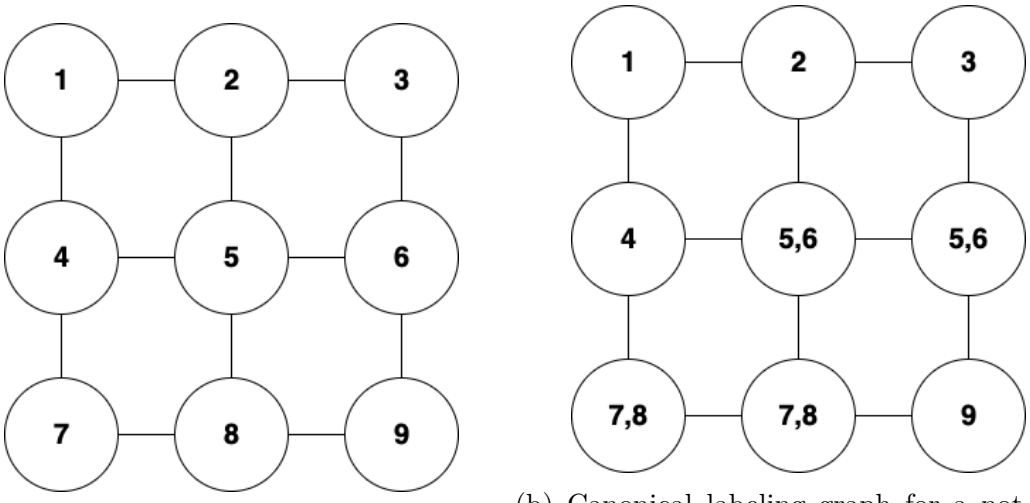
As part of this work, a version of McKay's algorithm, based on the description in 1980 [6], was implemented from scratch as well as the modified degree similarity approach (in Python 3). These two algorithms are part of the same implementation, except for the refinement procedure, which can either have a shatter based on identical degrees or degree similarity, δ_d. We have also added other parameters and functionalities to this implementation, which are covered in this section.

3.2.1 Pruning

The pruning procedure implemented for the degree based canonical labeling algorithms is based on its description in Chapter 2, where the search tree is pruned based on automorphisms and the binary representation's prefix.

The implementation maintains one canonical labeling for each different binary representation found and also keeps track of the largest binary representation, so it can compare with the prefix of the binary representation of nodes that are not discrete, in the middle of the search tree.

Our implementation has an improvement in the generation of the binary representation's prefix since it uses more than just the singleton cells from a partition to generate the prefix. To illustrate how this is done, consider the graph in Figure 3.6a and partition $\pi = [[1], [2], [3], [4], [5, 6], [7, 8], [9]]$. What the algorithm knows about this partition is that: node 1 will be mapped to 1, node 2 will be mapped to 2, node 3 will be mapped to 3, node 4 will be mapped to 4, node 5 will be mapped to 5 or 6, node 6 will be mapped to 5 or 6, node 7 will be mapped to 7 or 8, node 8 will be mapped to 7 or 8 and node 9 will be mapped to 9. So, the canonical labeling graph looks like the graph in Figure 3.6b.



(a) A simple 3x3 grid.

(b) Canonical labeling graph for a not discrete partition.

Figure 3.6: Example of prefix based pruning on a non discrete partition.

If the prefix was generated based only on singleton cells, it would only generate the first 4 bits of the adjacency matrix (0101) since node 5 does not belong to a singleton and, hence, its position in the adjacency matrix is not well defined. However, for this example, there are only two positions for node 5 and none of them is adjacent to node 1. Therefore, we can include one more bit in the prefix and write the 5th bit of the string as 0. The same happens for nodes $6, 7$ and 8 since independent of their position in the given partition they are not neighbors of node 1. Thus, it is possible to generate the complete first line of the adjacency matrix as 010100000. The algorithm that generates the prefix then goes on and tries to generate the second line of the adjacency matrix: node 2 is connected to nodes 1 and 3, but it is not clear if it is connected to nodes 5 or 6, as this depends on their final position. Therefore, the first four bits of the second line of the adjacency matrix can be written and included in the prefix, which is 1010 (the prefix only receives 10, the last two bits, since it is a representation of the superior triangle of the adjacency matrix). Finally, the binary representation's prefix of this partition is the concatenation of the lines that could be defined, and in particular, 1010000010. Note that this is much longer than the prefix that uses only singleton cells which is 101. The more bits the prefix has, the easier it is to prune the tree.

3.2.2 Other Parameters

We introduce a cell ordering parameter that determines whether the cells within a partition are ordered ascending or descending by degree. Suppose there are two cells c and d that are being examined by the refinement procedure and there are nodes from c with different number of neighbors in d, such that shattering will happen.

The cell c is split in two or more cells c_1, c_2, ..., c_s. McKay's algorithm orders the new cells ascending by the number of neighbors the nodes have in the cell that shattered it, and adds them to where the cell c was in the partition. So if $s = 3$ and each node from c_1 has 2 neighbors in d, each node from c_2 has 3 neighbors in d and each node from c_3 has 1 neighbor in d, then the order would be c_3, c_1, c_2.

Our algorithm allows the user to choose if cells are ordered ascending or descending within the partition. This is interesting when considering similar graphs and the fact that the nodes with the larger degrees will not change position in the ordered degree sequence even after some edges were removed from the graph. In fact, we evaluate this property in Chapter 5, where all our implementations are executed with descending order while nauty is executed as-is.

Chapter 4

Distance Based Canonical Labeling

Two fundamental structural information of graphs are node degrees and distances. Recall that distances are also preserved in the corresponding node pairs of isomorphic graphs. Thus, this information can be leveraged in solving the problem of identifying the structural similarity of two graphs. It is already clear that degrees and the adjacency matrix change when edges are removed or added. But what about distances and the distance matrix? Consider a real graph, as a social network for instance. There are usually lots of different paths connecting two nodes with the same length. Take Facebook, for example: two friends are likely to have multiple friends in common. Take two people from a social network that are not connected, but are at distance two (which means they have at least one mutual friend). If the edge connecting one of those nodes to the mutual friend is removed, it is likely that the two people have another friend in common and continue having distance two. This kind of redundancy can be helpful when removing edges from a graph. Also, the well-known principle of six degrees of separation which states that all people are within six social connections from each other [39], is another powerful evidence when considering redundancy of distances on real networks.

However, when a single edge is removed, at least one distance from the distance matrix will change: the one that represents the nodes adjacent to the removed edge and, hence, at distance one. But, on real networks, chances are that this distance will become two since the nodes probably are part of some triangle. Moreover, since real networks are likely to have path redundancy, all shortest paths that used this specific edge will likely be substituted by another path that has the same length as before, namely another shortest path. This redundancy will be exploited when using distances in the algorithm soon to be presented. Moreover, to consider the fact that distances 1 will likely change to 2, they will be considered equivalent from the algorithm's perspective.

The algorithm proposed in this dissertation is based on McKay's canonical labeling algorithm [6], covered in detail in the last two chapters. Its purpose is to solve both the isomorphism problem and the graph matching problem. The idea is for the canonical labeling algorithm to tackle the graph matching problem while also solving the isomorphism problem. The search tree process is identical to McKay's algorithm, the difference is on the refinement procedure (it doesn't use degrees), the pruning method and the way of encoding a graph into a string. Remember that McKay encodes a graph using the binary representation of the superior triangle of its adjacency matrix, and uses this information to decide which graph to return. This does not work well in the graph matching problem since two graphs with different, but similar, adjacency matrices may have different representations, while their representation should be similar (or identical) to have a similar canonical labeling and, thus, determine whether or not the graphs are similar.

4.1 Distance Refinement Procedures

This work defines two different refinement procedures using only distance information. Both methods receive a partition π and return a coarsest partition π', which is finer than π and is also an equitable partition. Although the previously given definition of equitable partition requires that every node in a cell has the same number of neighbors on every other cell, for all cells, the definition of equitable partition is slightly different for each distance refinement procedure presented. It is important to point out, however, that both procedures are based on McKay's degree refinement procedure [6] and the main difference is the shattering of the cells since it does not use degree or neighborhood information.

4.1.1 Distance

The idea of this procedure is to shatter cells based on the minimum path length between a node and a cell. The minimum path length between a node u and a cell C is the smallest path length between u and the nodes in C.

Definition 4.1.1. *A partition π is equitable in the distance procedure if:*

$$dist(u, c_j) = dist(v, c_j), \forall u, v \in c_i, \forall c_i, c_j \in \pi$$

In other words, if for every pair of cells c_i and c_j from π, every node w from c_i has at least one node from c_j with distance d and all other distances to the other nodes from c_j are larger or equal to d. Once again, distances with values 1 and 2 are considered the same value (distances 2 are overwritten and have value 1).

Also, this procedure does not use the distance between a node and itself, which is 0, to calculate the distance between a node and its own cell. In particular, $dist(v, c_i) = \min_{v \in c_i \setminus u} dist(u, v)$. The reason is that for any partition π, the shatter would have to ignore all pair of cells $(c_i, c_i), 1 \leq i \leq |\pi|$, since all nodes from c_i would have distance 0 to the cell c_i and there would be nothing to split. This should be avoided to accelerate the splitting process and, hence, the algorithm.

However, the initial partition, the one with a single cell containing all nodes, must be treated differently since the minimum distance between all nodes and the cell will be 1 (considering there are no isolated nodes in the graph). Thus, for the first split only, the eccentricity [1] of each node is used as their distance feature. All nodes with the same eccentricity are placed in the same cell. Once again, the cells are ordered based on this value and a parameter decides if they are ordered ascending or descending.

For example, consider the graph in Figure 4.1. The algorithm starts with the initial partition $\pi = [[1, 2, 3, 4, 5, 6, 7]]$ and proceeds to split the cell using the eccentricity of each node. Since nodes 3 and 5 have eccentricity 2, nodes $2, 4$ and 6 have eccentricity 3 and nodes 1 and 7 have eccentricity 4, it splits this cell in three new cells, $c_1 = [3, 5], c_2 = [2, 4, 6]$ and $c_3 = [1, 7]$, considering the algorithm is ordering the cells ascending. The next steps are all using the minimum distance between a node and a cell until the partition becomes equitable. It starts by taking the pair of cells (c_1, c_1), and calculates the minimum distance between each node from c_1 to the nodes in c_1. Since those distances are the same, nothing is split. Then it considers the pair of cells (c_1, c_2), following the same procedure. This continues until an equitable partition is reached. In this example, the partition $[[3, 5], [2, 4, 6], [1, 7]]$ is equitable and, thus, is returned by this refinement algorithm.

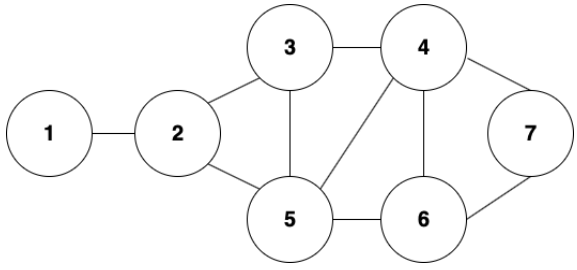


Figure 4.1: Graph with 7 nodes to exemplify distance based refinement.

4.1.2 Distance Vector

The previous refinement procedure used just the minimum distance between a node and a cell to shatter the cell. More information could be used to improve the

[1] The eccentricity of a node is the maximum distance between this node and every other node in the graph

shattering and in particular, all distances between a node and the nodes of a cell could be used.

This procedure to shatter cells is called distance vector. When analyzing a pair of cells (c_i, c_j), a distance vector for each node of c_i is constructed where each element is the distance to a node in c_j and the vector is ordered. Once again, distances with values 1 and 2 are identical (distances 2 are overwritten with value 1) and distances 0 are added to the vector here.

For example, consider the graph in Figure 4.1. The algorithm starts with the initial partition $\pi = [[1, 2, 3, 4, 5, 6, 7]]$ and proceeds to split this cell based on distance vectors, i.e., nodes with identical ordered distance vector stays on the same cell. First, all distance vectors are generated. For node 1: it has distance 0 to itself, distance 1 to node 2, distance 2 to nodes 3 and 5, distance 3 to nodes 4 and 6 and distance 4 to node 7, so its ordered distance vector towards this cell is $[0, 1, 1, 1, 3, 3, 4]$. The same procedure is done for all other nodes in the cell. The resulting ordered distance vector for each node is shown in Table 4.1. The next step of the algorithm is to split nodes according to their ordered distance vectors. An ordering between the ordered distance vectors must also be established and the values of the ordered vectors are used to establish the comparison. In particular, consider two ordered distance vectors v_1 and v_2 of identical length k. We say v_1 is smaller than v_2 if $\ni i, 1 \leq i \leq k$ such that $v_1[i] < v_2[i]$. Therefore, nodes 3 and 5 have the same vector, which is also the smallest vector, thus a new cell $[3, 5]$ is created and is the first in the new partition, that is given by $\pi' = [[3, 5], [2, 4, 6], [7], [1]]$ after the first iteration of the refinement procedure.

Node	Distance vector between node and cell
1	$[0, 1, 1, 1, 3, 3, 4]$
2	$[0, 1, 1, 1, 1, 1, 3]$
3	$[0, 1, 1, 1, 1, 1, 1]$
4	$[0, 1, 1, 1, 1, 1, 3]$
5	$[0, 1, 1, 1, 1, 1, 1]$
6	$[0, 1, 1, 1, 1, 1, 3]$
7	$[0, 1, 1, 1, 1, 3, 4]$

Table 4.1: Distance vectors example.

The distance vector refinement is finer than the distance refinement since it splits cells more rigorously and, hence, generates more cells, which makes it faster. However, it can end up breaking cells that should not be separated when trying to match two similar graphs. Thus, distance refinement may have an advantage in terms of generating a better canonical labeling. In fact, Chapter 5 shows the results

for a series of different graphs comparing these two approaches.

4.2 Canonical Labeling Representation

As in McKay's algorithm, the proposed distance based algorithm uses the graph's representation to determine the discrete partition, or canonical labeling, for a graph. The pruning procedure also uses this information to determine any possible pruning of the search tree. However, since the algorithm is based on distances, it makes sense to consider a representation that also uses distances. First, let's recall the binary representation used by McKay's algorithm, which will also be used in this algorithm to look for automorphisms.

4.2.1 Binary Representation

The binary representation of a canonical labeling is a string of 0's and 1's that uniquely represents a graph. It is simply the superior triangle of the adjacency matrix of the graph given by the canonical labeling, but without the main diagonal (since there are no self-loops in the graphs there is no need to consider the main diagonal which is always 0).

However, this representation is not interesting when working on similar but not identical graphs, since their representation will be different even when they differ by a single edge, and depending on the edge, their representation can be really different. Recall that the algorithm chooses the largest binary representation between all possibilities, so if the edge removed represents one of the most significant bits of the representation, the representation of the graph without an edge will be way smaller than the representation of the original graph. This will probably make the algorithm choose different representations for each graph, which will end up generating different canonical labelings for each graph and not identifying them as similar.

4.2.2 Distance Representation

The distance representation of a canonical labeling is a string containing natural numbers. It is composed of the superior triangle of the distance matrix of the graph that represents the canonical labeling but without the main diagonal. However, whenever there is a distance 2 in the matrix, the distance representation is given by the number 1 in that position. The goal is to leverage the same observation described earlier in this chapter, which assumes a real network has redundancy and, thus, when an edge is removed, the nodes adjacent to it will have distance 2 in the modified graph.

Although every graph has a unique distance matrix, since we are overwriting distances 2 with 1, it is possible that two different graphs (i.e., different edge sets) have the same distance representation. Thus, if two different canonical labelings have the same distance representation, then their respective binary representations are generated. If they are the same, then the canonical labelings represent the same graph (since each graph has a unique binary representation and vice versa). Likewise, if the binary representations are different, the graphs are different.

As with McKay's algorithm, the distance based algorithm will also return the canonical labeling associated with the largest distance representation of the search tree. However, since the largest distance representation may appear in different canonical labelings, they are all returned by the algorithm.

4.3 Pruning

The pruning procedure in the distance based canonical labeling is the same as the one in the degree based canonical labeling: it uses automorphisms to prune the tree and, for every non-terminal node, it generates a prefix for the distance representation of the partition associated to the node and compares with the largest distance representation yet found. The algorithm needs to keep track of all binary representations, the largest distance representation and the binary representations associated with this distance representation.

The algorithm runs without any pruning until the first terminal node is found, using its distance representation as the largest and recording its binary representation. From this point forward, whenever a node t of the search tree (terminal or not) is analyzed, there is a possibility that the search tree is pruned. Following we discuss all possible scenarios.

4.3.1 t is not a terminal node

If t is not a terminal node, then the corresponding partition is not discrete. The first step is generating the prefix of the binary representation associated with this partition. If the graph has automorphisms and, by trying to generate the prefix it ends up with the whole binary representation, the algorithm checks if this binary representation is identical to one already seen. If it is, then t is pruned from the search tree. If not, then the algorithm generates the prefix of the distance representation and it compares with the prefix of the largest distance representation seen so far (both prefixes need to have the same length). If the current node's prefix is smaller than the largest one, t is pruned from the search tree. Otherwise, the algorithm continues to search from node t.

4.3.2 t is a terminal node

If t is a terminal node, then the corresponding partition is discrete and the algorithm can generate the full binary and distance representations.

If the binary representation generated is identical to any of the binary representations generated so far, then the algorithm prunes the search tree based on automorphism just as in the case of degrees: it finds the deepest common ancestor a between t and the other terminal node u that has the same binary representation, and it removes the tree rooted at the child of a that is also ancestor of t.

If the binary representation of t is new, the algorithm saves it and proceeds to compare its distance representation d_r with the largest distance representation seen so far, $d_{\tau'}$, where τ' is the corresponding canonical labeling. If $d_r > d_{\tau'}$, then the largest distance representation becomes the representation of t. If $d_r < d_{\tau'}$, then nothing happens. If $d_r = d_{\tau'}$, then the two different graphs (since the binary representations are different) have the same distance representation and the next step depends on the problem being attacked. If the problem is isomorphism, then the algorithm just chooses the canonical labeling corresponding to the largest binary representation to be one corresponding to the largest distance representation node yet found. If the problem is graph matching, then all canonical labelings with the same largest distance representation are returned by the algorithm. Since the binary representation of two very similar graphs can be very different, the idea is to avoid using this information to decide the canonical labeling that will be returned in this case. At the end of the algorithm, if the largest distance representation has more than one canonical labeling associated with it (each one having a different binary representation), all such canonical labelings are returned.

These multiple canonical labelings can then be used to identify a good correspondence between the node set of two similar graphs. In particular, consider two graphs G_1 and G_2, all canonical labelings generated for G_1 and all canonical labelings generated for G_2, which can be used to find the best correspondence.

4.4 Application to Graph Matching

Unlike the binary representation, different graphs (in terms of their edge set) can have the same distance representation since such representation does not encode the exact distance matrix, but a modification where distances two are overwritten with one. Since the idea of the distance based canonical labeling algorithms is to return the canonical labeling associated with the largest distance representation found in the search tree, it can return multiple canonical labelings representing different graphs that all have the same largest distance representation.

Note that if the focus is on the isomorphism problem, then just the canonical labeling with the largest binary representation among all canonical labelings associated with the largest distance representation needs to be returned by the algorithm. This single canonical labeling represents a single graph and is the exact same for the whole isomorphism class of the graph given as input.

However, if the objective is to tackle the graph matching problem, all canonical labelings associated with the largest distance representation should be returned for each graph. Recall that in this case the two graphs are not isomorphic, which means they have different edge sets and thus the edge sets of the graphs permutated by the resulting canonical labelings will also be different. The multiple canonical labelings of the two graphs can then be used to find the best match between the edge sets of the two graphs. The idea is to choose the pair of canonical labelings, one for each input graph, which minimizes the differences in their edge set. The error function $e(\tau_1, \tau_2)$ compares two canonical labelings τ_1 and τ_2 of the graphs G_1 and G_2 and is given by:

$$e(\tau_1, \tau_2) = \frac{(\sum_{e \in E_{\tau_1}} \mathbb{I}(e \notin E_{\tau_2})) + (\sum_{e \in E_{\tau_2}} \mathbb{I}(e \notin E_{\tau_1}))}{|E_{\tau_1}| + |E_{\tau_2}|}$$

In other words, $e(\tau_1, \tau_2)$ is the number of edges that are in one graph permutated by the canonical labeling but not in the other. In particular, consider two graphs G_1 and G_2, and let $L_{\tau_{G_1}}$ and $L_{\tau_{G_2}}$ denote the lists containing the canonical labelings of G_1 and G_2, respectively. The error $e(\tau_1, \tau_2)$ is calculated for all pairs of canonical labelings formed by one canonical labeling from $L_{\tau_{G_1}}$ and one from $L_{\tau_{G_2}}$. The pair that minimizes $e(\tau_1, \tau_2)$ is then returned as the canonical labelings for the graphs G_1 and G_2. In particular, $\tau_1^*, \tau_2^* = \text{argmin}_{\tau_1 \in L_{\tau_{G_1}}, \tau_2 \in L_{\tau_{G_2}}} e(\tau_1, \tau_2)$, where τ_1^* and τ_2^* are used to determine the corresponding nodes in G_1 and G_2.

4.5 Theoretical Results

Two theoretical results concerning algorithm correctness for the distance based canonical labeling are provided, using both distances and distance vectors to shatter the cells and partitions. First, the algorithm produces a canonical isomorph function G_τ to an input graph G, which represents the whole isomorphism class of G. Second, the algorithm can recover the entire edge set of two different graphs if the distance matrices of the two graphs are the same, except values that are 1 in one matrix and 2 in the other.

Definition 4.5.1. *Let D_{G_1} and D_{G_2} denote the distance matrices for G_1 and G_2, respectively. Then the distance canonical labeling algorithms will always be able to*

recover the entire edge set G_1 and G_2 (except the missing edges) if their distance matrices respect the following condition:

$$(D_{G_1}(i,j) = D_{G_2}(i,j)) \vee (D_{G_1}(i,j) = 1 \wedge D_{G_2}(i,j) = 2)$$
$$\forall i, j \in V$$

We call this the distance condition.

Although this condition was defined assuming the two graphs G_1 and G_2 were not previously anonymized (and the original labels of the nodes are the same on both graphs), it is still true if the graphs are anonymized. The only difference is that now the process does not know where two nodes from G_1 are on G_2 and, hence, does not know the distance between these nodes on G_2 and how to compare them. However, it still possible to check the condition: if G_2 has a permutation γ on its nodes, creating a graph G_2^γ, such that the distance condition is satisfied, we can say that G_1 and G_2 satisfies the distance condition and the distance algorithms will be able to recover the entire edge set G_1 and G_2 (except the missing edges). With that, the distance algorithms work the same way whether or not the graphs are previously anonymized, since they are based on the graphs' structures and do not make use of the nodes' original labels.

The theoretical results will be presented following the proof arguments of McKay's algorithm correctness [6, 22].

4.5.1 The Isomorphism Problem

The first step is proving the correctness of the algorithm for the isomorphism problem following the same arguments as the degree based algorithm (not our implementation of degree similarity). In other words, we present a theoretical proof that the proposed distance based canonical labeling algorithms produce a canonical isomorph function G_τ to an input graph G, which represents the whole isomorphism class of G.

Lemma 4.5.1. *The equitable partition π' returned by the refinement algorithm on any partition π and a graph G is finer than π.*

Proof. Each iteration of the refinement algorithm tries to shatter some cell V_i, producing a new ordered partition whose length is greater than the length of π. Since a partition has a length at most n, the number of iterations in the refinement algorithm is bounded and it is certain to terminate.

The algorithm only stops when an equitable ordered partition is obtained (and every discrete partition is equitable). At each iteration the current partition is finer than the one from the previous iteration and, hence, is finer than the input partition π, so the equitable partition π' produced is also finer than the input partition π.

□

Lemma 4.5.2. *Let G and H be two graphs and γ a permutation of the nodes of G such that $G^\gamma = H$. Then $T(H) = (T(G))^\gamma$, where T is the search tree induced by the distance based algorithms. In particular, every node v_H in $T(H)$ has an equivalent node v_G^γ on $T(G)$, and vice-versa.*

Proof. The refinement algorithm respects the action of the permutation function γ, i.e., if π'_G is the refinement of π_G and $(\pi_G)^\gamma = \pi_H$, then the refinement of π_H, namely π'_H, is equal to $(\pi'_G)^\gamma$. For example, consider the first partition, the one with a single cell containing all n nodes from G. Using distances, the first step is to split the nodes based on their eccentricity and order the cells based on each cell's eccentricity value. The resulting partitions π_G and π_H from both graphs will have the same number of cells and each cell V_i from π_G, with $1 \leq i \leq |\pi_G|$, will have the same number of nodes from the respective cell W_i from π_H. The difference will be on the nodes' labels on each cell respecting the permutation function, so $(\pi'_G)^\gamma = \pi'_H$.

This argument given for the shattering based on eccentricity is also valid for using distances or distance vectors.

Once an equitable partition π' is obtained, the next step is to generate its children, to continue the execution of the algorithm. In particular, the first nontrivial cell from π' will be taken apart. If the first nontrivial cell of π_G is V_i, then the first nontrivial cell of π_H will be W_i, which is V_i^γ. So the same cell i, on both partitions, will be the one chosen to generate the children in the search tree. The algorithm generates all possible children from the equitable partition since it takes each node from the chosen cell and creates a new cell with this node, maintaining the remainder of the partition. Figure 4.2 shows two equitable partitions $\pi_1 = [[1, 3, 5, 7], [2, 4, 6, 8], [5]]$ and $\pi_2 = [[A, B, C, D], [E, F, G, H], [I]]$, with π_2 being a permutation of π_1. The chosen cell to be taken apart is the same on both partitions: the first one. The children from π_1 and π_2 are the same under the permutation function between π_1 and π_2.

Figure 4.2: Example of children of two isomorphic equitable partitions.

□

Theorem 4.5.3. *G_τ is a canonical isomorph function*

Proof. In the search tree, the ordered partition π associated with a node is strictly finer than the ordered partition associated with its parent. Hence, the search tree $T(G)$ is finite and the number of leaves (that represent the discrete partitions) is also finite, and the algorithm terminates. The output G_τ, by definition, is an isomorph of the input graph G.

By Lemma 4.5.2 every leaf node p with discrete partition π in $T(G)$ has a correspondent in $T(H)$, a leaf node p^γ with discrete partition π^γ. So, the set of isomorphic graphs produced by the leaf nodes of $T(G)$ is the same set of isomorphic graphs produced by the leaf nodes of $T(H)$. Each discrete partition is associated with its distance representation and its binary representation. The sets of distance representations are the same for both graphs G and H and the algorithm chooses the largest distance representation from these sets. In case there is more than one discrete partition with the largest distance representation, the algorithm chooses the one with the largest binary representation (same binary representations means that the graphs are the same, namely an automorphism). Therefore, the algorithm chooses the same discrete partition in $T(G)$ and $T(H)$, meaning that the edge set of G_τ will be the same edge set of H_τ.

Even when there is pruning on the search trees, the theorem still applies since the idea is that G_τ is the largest distance representation (if there is a tie between the nodes, then the largest binary representation is used) and the pruning only removes nodes from the tree that have a smaller distance representation from the largest one found yet (so nodes with smaller representations would never be chosen) or if there are automorphisms in the graph (but, in this case, the specific canonical labeling returned does not matter since the graphs are identical).

□

4.5.2 The Graph Matching Problem

Here, the two graphs G_1 and G_2 are not isomorphic and G_2 is a copy of G_1 after randomly removing some edges. Also, recall that the distance condition defined requires that the distance matrices of G_1 and G_2 are the same except for some distances that are 1 in G_1 and 2 in G_2.

Lemma 4.5.4. *If two graphs G_1 and G_2 respect the distance condition, then the edge set of the canonical labeling of G_2 will be the same edge set of the canonical labeling of G_1 except the missing edges (removed from G_1 to generate G_2).*

Proof. The theoretical proof is pretty much the same given for the isomorphism problem. Lemma 4.5.1 is still valid since it only states about the refinement procedure returning a finer partition then the one received. Lemma 4.5.2 is also valid as

is, since the partitions will be shattered the same way on both graphs and, by the same proof, the search tree for both graphs will be isomorphic.

The only detail that needs a little more attention is in Theorem 4.5.3. Now, instead of G_τ being the canonical isomorph function and, hence, representing the whole isomorphism class of G, G_τ represents the isomorphism class of all graphs isomorphic to G under the distance condition. The proof is the same given above, with a small difference: when there are two terminal nodes with the same distance representation and different binary representations, hence, representing different graphs, the algorithm does not choose one node to continue, but it maintains both representations. At the end of the execution, if the largest distance representation found happens to have more than one canonical labeling associated with it, all canonical labelings are returned. What is important here is saying that the same thing will happen on both graphs, based on the same proof given on Theorem 4.5.3. Since both graphs will output the same set of canonical labelings, all canonical labelings returned for one graph are compared to all canonical labelings returned for the other graph using the error function $e(\tau_1, \tau_2)$ (see section 4.4), returning the pair of canonical labelings that minimizes it. Since $e(\tau_1, \tau_2)$ is basically a sum of the edges that are on one graph but not the other, the pair that minimizes it will be the one where only the edges removed from G_1 are missing in the canonical labeling of G_2.

□

If the graphs are similar but don't satisfy the distance condition, the algorithm might still be able to correctly match the two graphs, but there is no theoretical proof behind it defined yet. Chapter 5 shows the results on many different graphs, where this is observed sometimes.

4.6 Implementation

All distance based algorithms were designed and implemented using Python 3 and most of the methods and all parameters found in the degree based algorithms are also available in this implementation. The core of the four algorithms (our McKay implementation, degree similarity, distance and distance vector) is the same, with parameters stating the course of action in each variation.

There is an extra parameter that is used to determine which problem the algorithm is trying to solve, isomorphism or graph matching. This parameter leads to an important difference, both at the refinement algorithm and the generation of the distance representation. When the algorithm is trying to solve isomorphism, it does not consider distances 1 and 2 to be the same. Thus, the isomorphism problem

often runs much faster than the graph matching problem, for two reasons. The refinement is more powerful (it can break more cells); the pruning method is more powerful (it is easier to happen since in this case all different graphs have different distance matrices and, hence, different distance representations). Whenever the algorithm is working on a partition and it can generate a partial (or the whole) distance representation, it can compare with the largest representation found so far. The graph matching problem prunes less since different graphs can have the same modified distance matrix and, whenever a new canonical labeling is found, if it has the same partial (or whole) distance representation as the largest one yet found and a different binary representation from the ones found, nothing can be done.

Chapter 5

Evaluation

This chapter presents an evaluation of six different canonical labeling algorithms considering both real networks and random graphs models. The goal is to understand and characterize the performance of the algorithm in the task of graph matching. Therefore, the evaluation is performed by running the canonical labeling algorithm in two different but similar graphs G_1 and G_2 and comparing the graph structure of the resultant canonical forms $C_\tau(G_1)$ and $C_\tau(G_2)$.

5.1 Methodology

Six different canonical labeling algorithms are considered: the nauty package [24], pybliss (an official implementation of Bliss in Python), a customized implementation of McKay's algorithm as described in the original 1980 paper [6], degree similarity, distance and distance vector. All graphs used for evaluation ran on all the six algorithms.

This evaluation takes two graphs G_1 and G_2, where G_2 is a copy of G_1 after removing a small number of edges. The first step is generating the set of graphs that will pass through the canonical labeling algorithms. For each graph G_1 and a given amount of edges to be removed, G_2 is constructed by removing such edges from G_1 uniformly at random. Since G_2 is a random graph, this procedure is repeated 100 times, generating 100 graphs like G_2 for each original graph G_1. When considering a random graph model, i.e., graphs generated from a mathematical random graph model, then a hundred different G_1 for each parameter configuration are considered. Also, since two canonical labeling algorithms depend on the distance matrix, this matrix is generated for all graphs generated. The four algorithms developed for this work were executed ordering the cell and partitions in descending order.

Figure 5.1 exhibits the evaluation and the generation of G_2 for the six algorithms tested. The degree based algorithms return a single canonical labeling for each graph, so both L_{τ_1} and L_{τ_2} only contain one element. Meanwhile, the distance

based algorithms might return multiple canonical labelings, so the next step in this case is choosing one element from L_{τ_1} and one element from L_{τ_2}, that minimizes the error function presented on the last chapter, on section 4.4.

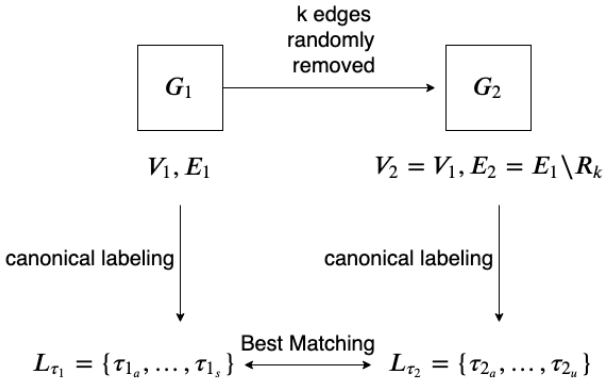


Figure 5.1: Evaluation of the graph matching problem.

5.2 Metrics

Different metrics are used to compare the resultant canonical labelings from the two input graphs G_1 and G_2, as well as other metrics to characterize the behavior of the algorithms.

5.2.1 Evaluation Metrics

Perfect Match

The perfect match metric is defined by the function $p(G_1, G_2)$, that takes two graphs G_1 and G_2 as input and returns either true (1) or false (0) as follows:

$$p(G_1, G_2) = \begin{cases} 1, & \text{if } E_\tau^2 \subseteq E_\tau^1 \\ 0, & \text{otherwise} \end{cases} \tag{5.1}$$

In other words, if the metric is true (1), then all edges from $C_\tau(G_2)$ are also edges in $C_\tau(G_1)$ and the only difference between the two sets are the edges from $C_\tau(G_1)$ that are not present in $C_\tau(G_2)$, that were removed from G_1 when generating G_2.

When $p(G_1, G_2) = 1$ the algorithm was able to identify that the graphs are similar and found the perfect graph matching between G_1 and G_2. Note that whenever the canonical labeling algorithm receives two isomorphic graphs, this metric will be true for the algorithms: distance, distance vector, our implementation of McKay, nauty and pybliss. Furthermore, if the graphs are not isomorphic but are similar and respect the distance condition, as defined in Chapter 4, the algorithms based on

distance will always be true. However, even when this is not the case, the algorithm might still be able to find a perfect match between the graphs.

Since each configuration considers 100 graphs, this metric is represented by the number of times it returned true divided by the total number of graphs.

Different Edges Mean

The different edges mean metric is defined by a function $d_e(G_1, G_2)$ that takes two graphs G_1 and G_2 as input and returns a number between 0 and 1, defined as:

$$d_e(G_1, G_2) = \frac{\sum_{e \in E_\tau^1} \mathbb{I}(e \notin E_\tau^2) + \sum_{e \in E_\tau^2} \mathbb{I}(e \notin E_\tau^1)}{|E_\tau^1| + |E_\tau^2|}, \qquad d_e(G_1, G_2) \in [0, 1] \tag{5.2}$$

In other words, this metric counts the number of edges that appear on one of the graphs but not on the other, and normalizes by the total number of edges (the sum of the edge sets of $C_\tau(G_1)$ and $C_\tau(G_2)$).

The algorithm calculates the percentage of different edges for each graph pair and takes the average and standard deviation across these pairs.

There are two variations regarding this metric: one is the average for all executions and one is the average for the executions where the distance condition is not respected. This variation is generated only for the distance based canonical labeling algorithms and measures their performance when they are not guaranteed to succeed.

Another important detail is that even when there is a perfect match between two graphs, this metric won't be 0 since, in this case, the number of different edges between the two graphs is equal to the number of edges removed from G_1. This is the smallest possible value for the numerator and greater than zero.

Different Nodes Mean

The different nodes mean metric, $d_n(G_1, G_2)$ is similar to the different edges mean, with the difference that it considers nodes, not edges. It is defined as follows:

$$d_n(G_1, G_2) = \frac{\sum_{i=1}^{n} \mathbb{I}(\tau_{G_1}(i) \neq \tau_{G_2}(i))}{n}, \qquad d_n(G_1, G_2) \in [0, 1] \tag{5.3}$$

In other words, this metric counts the number of nodes labeled differently in the two graphs. For example, consider two canonical labelings $[[1], [2], [3], [4]]$ and $[[1], [2], [4], [3]]$. In this case, there are 2 nodes mapped the same and 2 nodes mapped differently, so this metric would be 0.5. If this metric is 0, then there is a perfect match between the two graphs. However, if this metric is greater than 0, a perfect

match is still possible since this metric does not consider automorphisms (the edge set can be the same and the node labeling can be different).

As with the different edges mean, for all the configurations the algorithm calculates the percentage of different nodes and takes the average and standard deviation across all pairs of a given configuration. This is the value of the metric.

Also, there are two variations regarding this metric: one is for all executions and one is for the executions where the distance condition is not respected.

5.2.2 Robustness Metrics

Besides the evaluation metrics, metrics to characterize the graphs before and after some edges are removed have also been developed. The idea is to use these metrics to understand the robustness of the degree and distance algorithms to edge removal in different scenarios. These metrics help understand some of the results observed by the evaluation metrics.

Highest Degree Nodes

The idea of this metric is to understand the results of the degree based canonical labeling algorithms. Since the algorithms that use degree for the refinement procedure produce partitions that are, in a way, ordered by degrees, it is important to understand how the degrees change after some edges are removed from the initial graph. The goal is to determine the fraction of the nodes that will end up on the same position on the resultant equitable initial partition (that breaks the initial cell with all nodes based on their degrees) of the two graphs. This is indicative that these nodes might end up being correctly mapped into the same node on both canonical labelings.

The idea is to analyze the nodes with larger degrees since they are the more distinct ones. Usually, there are a lot of nodes with small degrees and a few nodes that have higher distinct degrees. Also, after some edges are removed, the nodes with the higher distinct degrees likely remain the same and this is exactly what this metric measures.

Definition 5.2.1. *Let V_1 denote the set of nodes in G_1 and V_2 the set of nodes in G_2, with $|V_1| = |V_2| = n$. The function $\Theta(V)$ returns a set of nodes ordered descending by their degrees. The highest degree nodes metric H is given as:*

$$H = \frac{1}{n} \max\{i | (\Theta(V_1)_i = \Theta(V_2)_i) \wedge (d(\Theta(V_1)_i)) \neq d(\Theta(V_1)_{i+1})) \wedge d((\Theta(V_2)_i)) \neq d(\Theta(V_2)_{i+1}))\} \tag{5.4}$$

The first step is ordering the nodes descending by degree for the two graphs. After that, the metric counts how many nodes have distinct degrees on its graphs and are on the same position on both lists, stopping after one of the conditions fails. Then, it is divided by the number of nodes to normalize the result.

For example, consider the Table 5.1 that shows two lists of ordered nodes and their degrees. There are 6 nodes and the highest degree nodes metric will be 0.5 since it will count the first three nodes and stop on the fourth node because they are different on the two lists.

$O(V_1)$	$d(O(V_1))$	$O(V_2)$	$d(O(V_2))$
A	12	A	10
B	10	B	9
C	8	C	8
D	7	E	7
E	7	D	6
F	5	F	5

Table 5.1: Highest degree nodes metric example.

Intuitively, graphs with a high value in this metric will have a better performance on the degree based canonical labeling algorithms.

Eccentricity

Here, the idea is to understand if eccentricity is robust to edge removal, i.e., if the nodes eccentricity are maintained after some edges of a graph are removed. This is important for the distance algorithm, where the initial partition break is based on eccentricity and this defines the course of the algorithm.

Definition 5.2.2. *Let V_1 denote the set of nodes in G_1, V_2 the set of nodes in G_2, with $|V_1| = |V_2| = n$, and $Ecc(V)$ denote the list containing the eccentricity value for the nodes. The eccentricity robustness metric $Ecc(G_1, G_2)$ is defined as:*

$$Ecc(G_1, G_2) = \frac{\sum_{i=1}^{n} \mathbb{I}(Ecc(V_1)_i = Ecc(V_2)_i)}{n} \tag{5.5}$$

In other words, it counts the number of nodes that have the same eccentricity on the two graphs and, to normalize, divides by the number of nodes.

Distances

The distance robustness metric is based on the same concept as the eccentricity metric: understand how the distances are maintained after some edges of a graph are removed.

Definition 5.2.3. *Let D_G represent the distance matrix of the graph G. The distance robustness metric $D(G_1, G_2)$ is defined as:*

$$D(G_1, G_2) = \frac{\sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbb{I}(D_{G_1}(i,j) = D_{G_2}(i,j))}{\frac{n(n-1)}{2}} \tag{5.6}$$

In other words, it counts the number of node pairs that have the same distance on the two graphs and, to normalize, divides by the number of node pairs.

To understand robustness of distances to edge removal we analyze the number of distances that have changed based on the amount it has increased (distances always stay the same or increase after removing edges from a graph). This metric is called k units distance increase and it represents the fraction of distances that have increased by more than k units, defined as follows:

$$D_k(G_1, G_2) = \frac{\sum_{i=1}^{n} \sum_{j=i+1}^{n} \mathbb{I}(D_{G_1}(i,j) - D_{G_2}(i,j) > k)}{\binom{n}{2}} \tag{5.7}$$

In other words, the number of distances that remain the same are not considered in this metric, the number of distances that have changed (no matter by how much) are considered in the $k = 0$ unit, the distances that increased more than 1 are considered in the $k = 1$ unit and so on. Since some edges are removed, $D_0(G_1, G_2)$ will always be greater than 0, accounting for at least the distances directly affected by the edges removed.

5.3 Results on Random Graph Models

This section presents the evaluation of the six canonical labeling algorithms on random graph models. In particular, the Watts-Strogatz [2] and the Erdős–Rényi [32] models are considered. The first is the small networks model and has many short cycles while the latter generates local tree-like structures, thus generating very different graphs.

5.3.1 Watts-Strogatz

The Watts-Strogatz model is a random graph generation model that generates graphs with small-world properties, like short distances, high clustering coefficient, and concentrated degrees. The model has 3 parameters: n is the number of nodes in the graph, k determines the initial degree, and p controls the randomness of the graph. The procedure starts generating a lattice graph with n nodes organized in a circle and adds edges to neighbors at distances k or less in the circle (thus, each node has initial degree $2k$). After that, each edge is repositioned with probability p. The new edge points are chosen uniformly between all nodes. The Figure 5.2

shows how the model works when increasing p. The first graph, on the left, with $p = 0$, is a lattice $2k$-regular graph. The last graph, on the right, with $p = 1$, shows a completely random graph, where all edges have been repositioned uniformly at random.

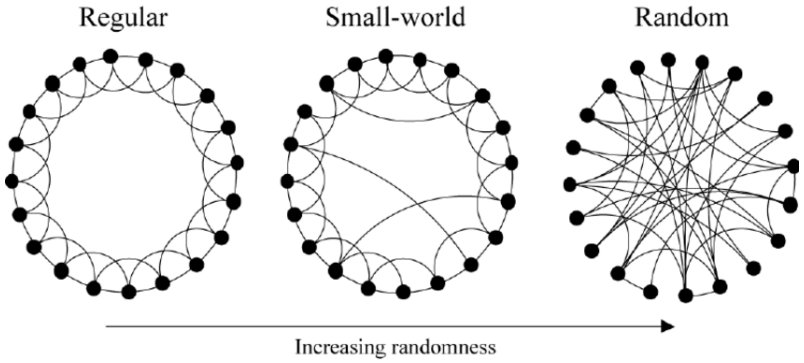


Figure 5.2: The Watts-Strogatz model with increasing p (extracted from [2]).

For the evaluation, the graphs generated have $p = 0.01$, $k = 8$ and $n = \{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1500\}$. For each value of n, a hundred graphs were generated. The amount of edges removed for this study was $e = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15\}$, $|e| = 11$. For each one of the 1100 graphs generated, there were 11 other graphs generated randomly removing edges accordingly to the values of e. In total, there were 12100 graph pairs passing through the six algorithms. The degree similarity algorithm was parameterized with $\delta_d = 1$ and $p = 0.1$.

Evaluation Metrics

Figure 5.3 shows the perfect match metric for all algorithms varying the number of edges removed from Watts-Strogatz networks with 1500 nodes. It shows that the perfect match ratio decreases monotonically with the number of edges removed for the two distance based algorithms and is always zero for all degree based algorithms. Moreover, with a single edge removal, the perfect match ratio is around 0.70 and 0.65 for the distance and distance vector algorithms, respectively. However, this ratio drops to around 0.10 for both algorithms when 5 edges are removed. Interestingly, all degree based algorithms failed in generating a single perfect match while in 70% of the cases where a single edge was removed the distance algorithm generated a perfect match. Note that the perfect match of the distance algorithms is very close to the distance condition being satisfied, shown by a specific curve in the plot, which represents the fraction of graph pairs that satisfy the distance condition presented in Definition 4.5.1. Recall that this fraction is a perfect match lower limit for the distance based algorithms, as stated in Chapter 4 by Theorem 4.5.4. For both distance algorithms, the performance observed is above this lower limit.

The behavior is similar when fixing the number of edges removed in 3 and varying the number of nodes, shown in Figure 5.4. The results do not indicate a clear trend as the number of nodes increases.

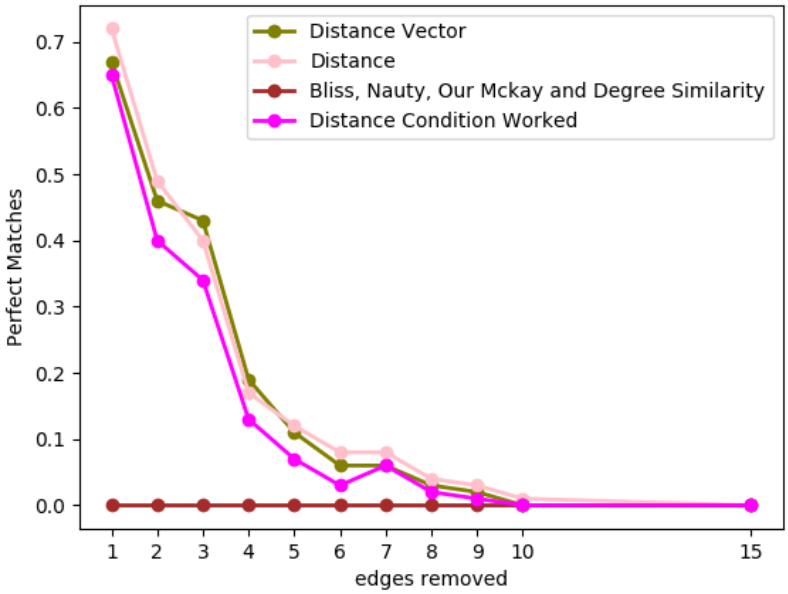


Figure 5.3: Perfect matches for WS networks with 1500 nodes.

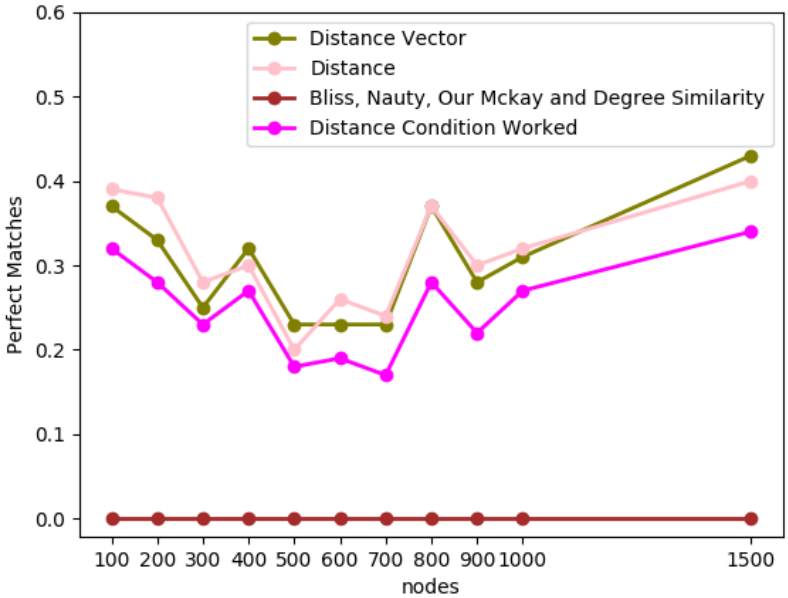


Figure 5.4: Perfect matches for WS networks when 3 edges are removed.

The different edges mean metric is shown in Figures 5.5, 5.6, 5.7 and 5.8 for different scenarios, where the first two ones show the results when fixing the number

of nodes in 1500 and varying the number of edges removed, while the latter ones show the results when fixing the number of removed edges in 5 and varying the number of nodes. Figure 5.5 shows the results for this metric for all algorithms being evaluated, each represented by a different curve, in all executions, just like Figure 5.7 does, while Figures 5.6 and 5.8 only show the results for the distance based algorithms for the executions where the distance condition between the two graphs was not satisfied.

Figure 5.5 shows that the different edges mean ratio increases monotonically with the number of edges removed for all algorithms. Moreover, it exhibits that the fraction of different edges is at least 0.1 smaller for the distance based algorithms in comparison to nauty and Bliss, and at least 0.5 smaller in comparison to the degree similarity and our implementation of McKay's algorithm. Moreover, with a single edge removal, the different edges mean ratio is around 0.05 for the distance based algorithms, 0.25 for Bliss, 0.37 for nauty and 0.75 for our implementation of McKay's and degree similarity, which shows that the fraction of different edges for the distance algorithms are five times smaller in comparison to the best performance shown by a degree algorithm.

Figure 5.7 shows that the different edges mean metric tends to converge as the number of nodes increases. It also exhibits that the metric is at least twice as smaller for the distance based algorithms in comparison to all four degree based algorithms.

Similar results can be observed for the different nodes mean metric, shown in Figures 5.9, 5.10, 5.11 and 5.12.

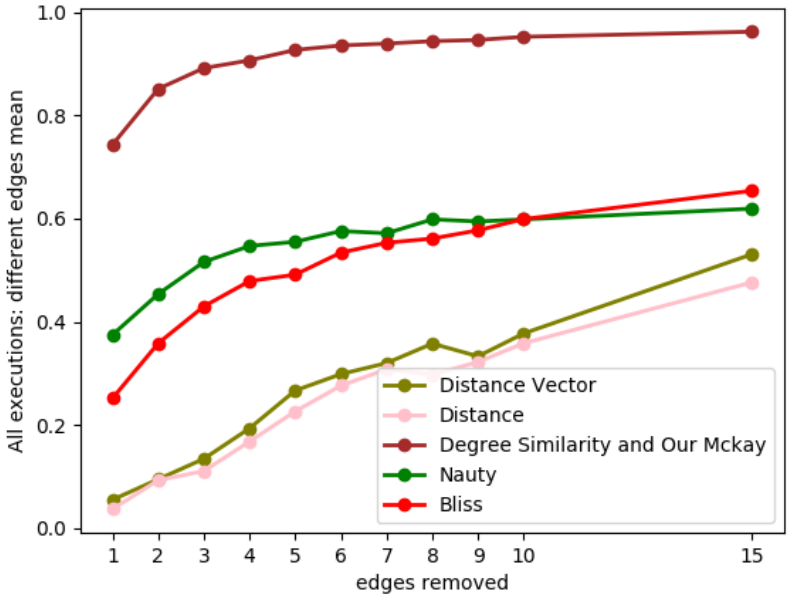


Figure 5.5: Different edges mean for WS networks with 1500 nodes.

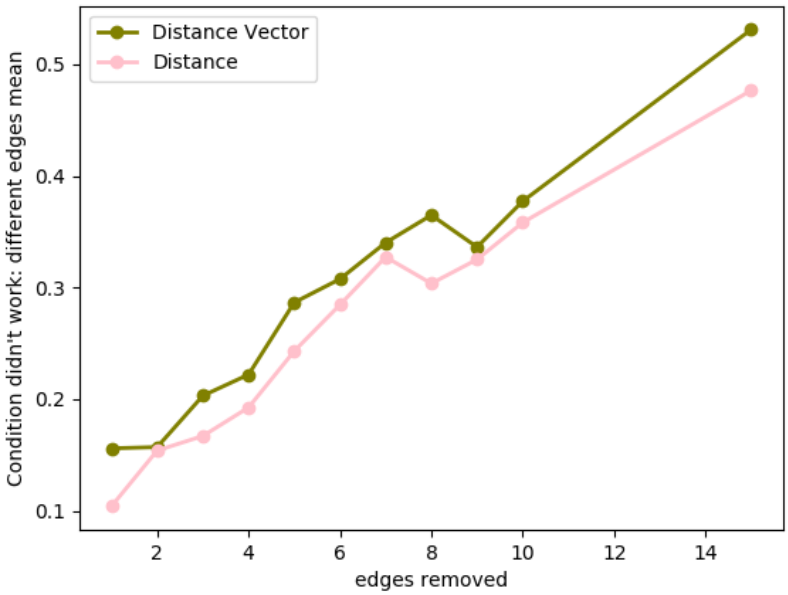


Figure 5.6: Distance condition not satisfied: different edges mean for WS networks with 1500 nodes.

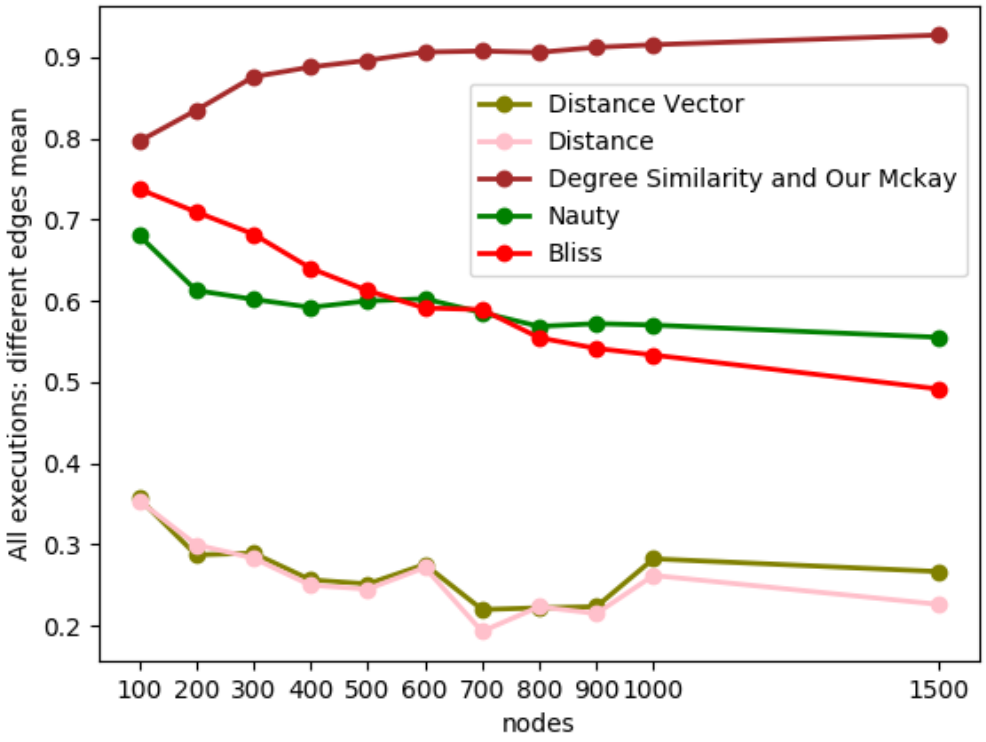


Figure 5.7: Different edges mean for WS networks when 5 edges are removed.

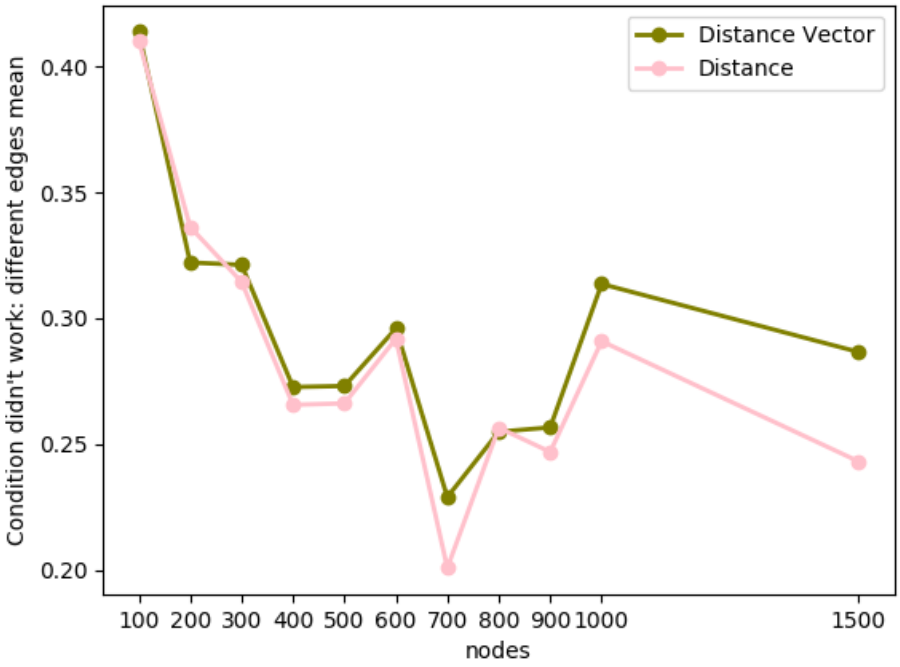


Figure 5.8: Distance condition not satisfied: different edges mean for WS networks when 5 edges are removed.

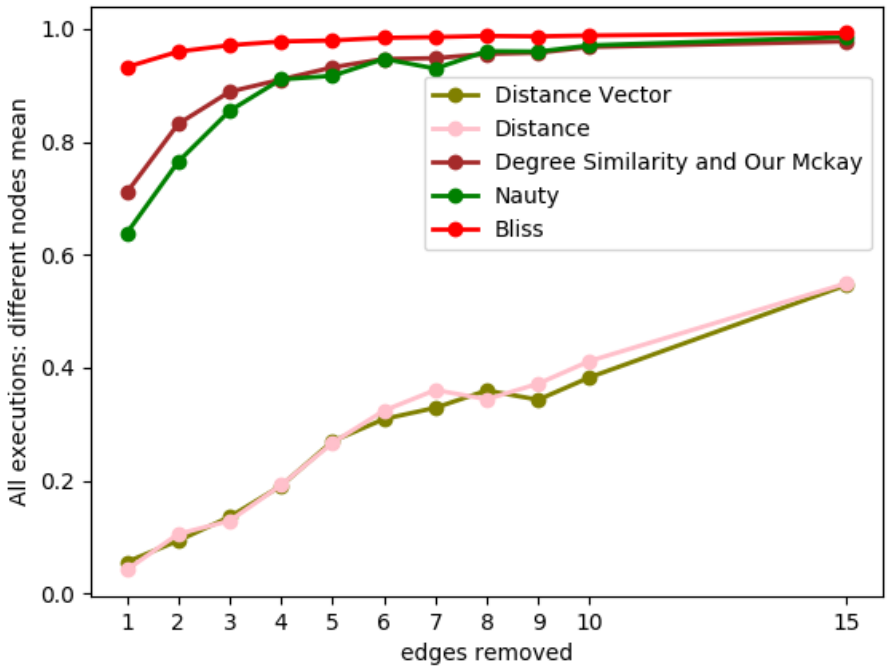


Figure 5.9: Different nodes mean for WS networks with 1500 nodes.

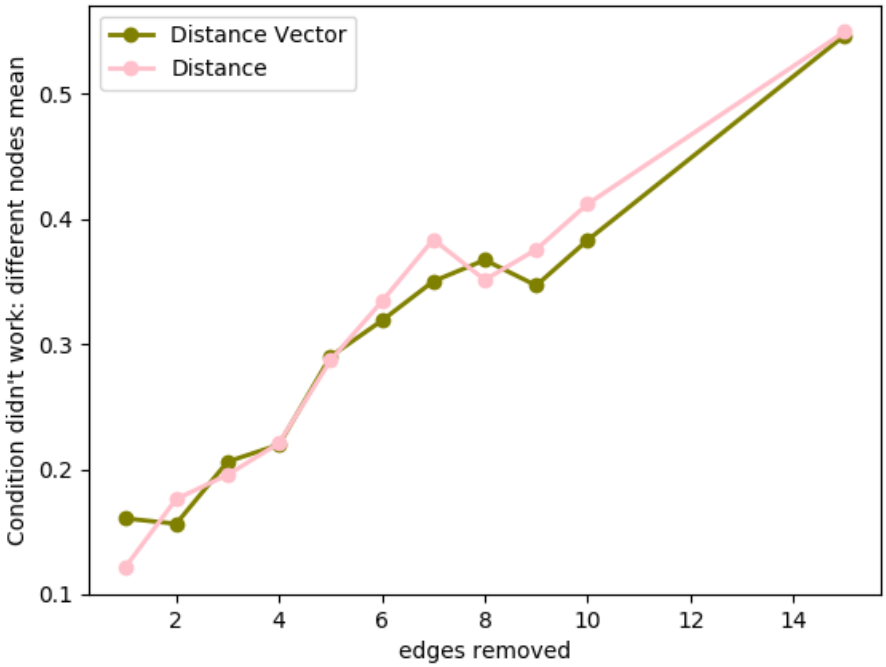


Figure 5.10: Distance condition not satisfied: different nodes mean for WS networks with 1500 nodes.

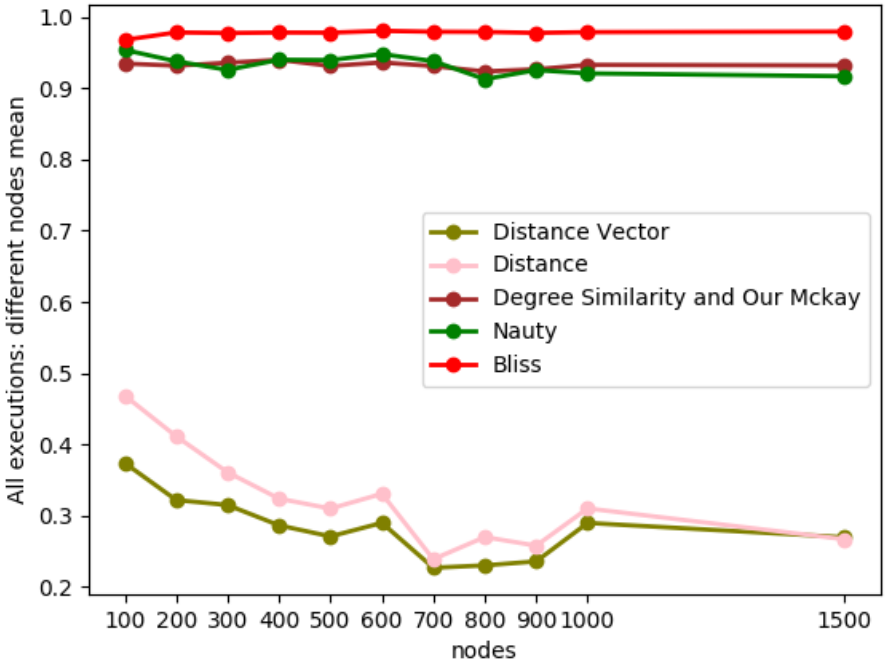


Figure 5.11: Different nodes mean for WS networks when 5 edges are removed.

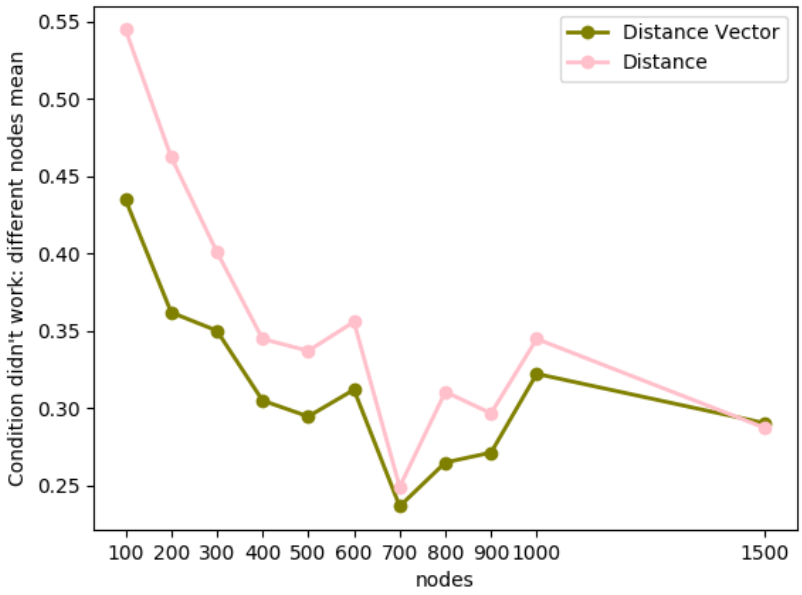


Figure 5.12: Distance condition not satisfied: different nodes mean for WS networks when 5 edges are removed.

Robustness Metrics

Table 5.2 shows the highest degree nodes metric for Watts-Strogatz networks with 1000 nodes. Note that the values are really small, all less than 0.05% of the nodes. This is one of the reasons why the degree based algorithms do not have a good performance.

Edges Removed	Mean	Standard Deviation
1	0.0004	0.0005
2	0.0004	0.0005
3	0.0003	0.0005
4	0.0003	0.0005
5	0.0003	0.0005
6	0.0004	0.0005
7	0.0004	0.0005
8	0.0004	0.0005
9	0.0003	0.0005
10	0.0003	0.0005
15	0.0003	0.0005

Table 5.2: Highest degree nodes values for Watts-Strogatz networks with 1000 nodes.

Figure 5.13 shows the eccentricity robustness metric as the number of edges

removed increases, where the curves represent the different network sizes. Note that the eccentricity metric has a similar trend on different network sizes as the edges being removed increase. In general, the results are positive for the distance based canonical labeling algorithm since the value of eccentricity robustness is small, below 4% when up to 4 edges are removed.

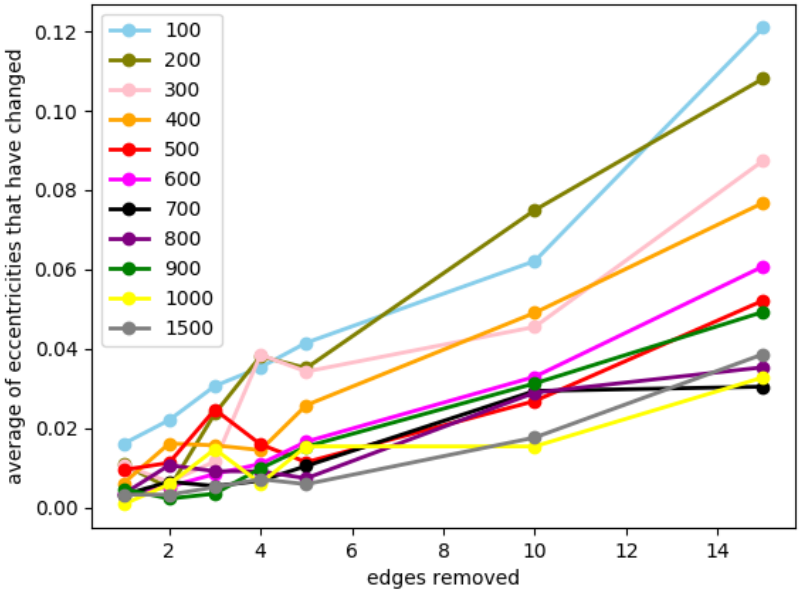


Figure 5.13: Eccentricity changed for WS networks.

Figure 5.14 shows the k units distance increase for each number of edges removed for a network with 500 nodes. The distances increase up to 19 units but G_2 is still connected. Another interesting observation is that more than 95% of the distance values are maintained even after 15 edges are removed. This positive result is one of the reasons the distance algorithms show good performance on this network. The results are similar to other network sizes of the WS model.

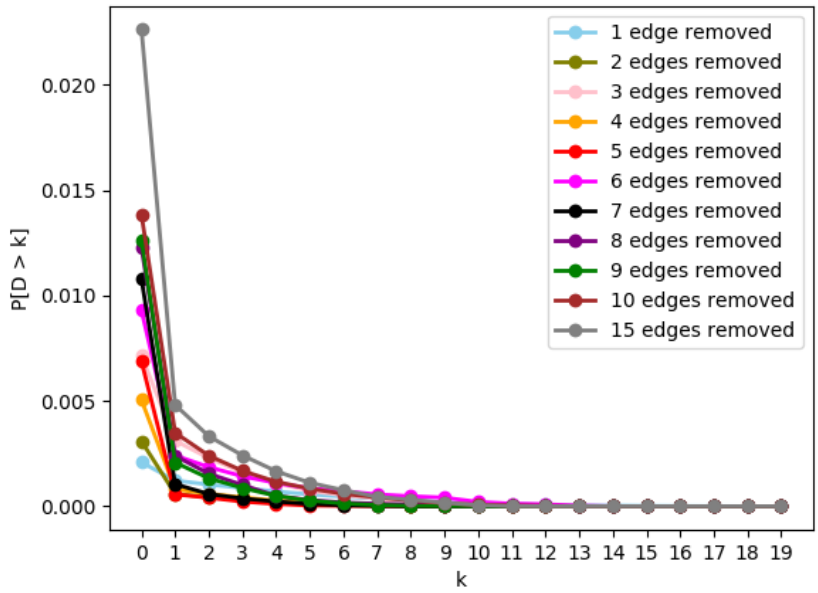


Figure 5.14: k units distance change for Watts-Strogatz networks with 500 nodes.

For all tested Watts-Strogatz networks, the two distance based canonical labeling algorithms outperform, in all metrics, all other algorithms.

5.3.2 $G(n, p)$

The Erdős–Rényi model [32], or $G(n, p)$ model, is the most traditional model for generating random graphs. It has two parameters: n is the number of nodes in the graph and p is the probability that an edge exists between a node pair. If $p = 0$, then there are no edges in the graph and all nodes are isolated. If $p = 1$, the graph is a complete graph. Thus, the larger p, the denser the graph, which means the average path length will be shorter, there will be a high clustering and the expected mean degree will be larger (since it is np).

This evaluation considers sparse graphs, with a small p, to generate graphs without much redundancy. For the tests, the parameters are $n = 100$ and $p = \frac{5}{n} = 0.05$. A hundred graphs with these parameters were generated and $e = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ edges were removed from each one, generating a new graph. In total, there are a thousand networks passing through each canonical labeling algorithm. The degree similarity algorithm had the parameters set to $\delta_d = 1$ and $p = 0.5$.

Evaluation Metrics

The perfect match metric is presented in Figure 5.15. Note that until 3 edges are removed, the distance algorithm outperforms all other algorithms, outperforming

even the distance condition being satisfied, which is always zero. However, the perfect match ratio decreases monotonically and at a very fast pace, becoming zero for all algorithms after four or more edges are removed. The results also show that the only other algorithm capable of finding a perfect match is the degree similarity, while all other algorithms, including the distance vector, can not find any perfect match regardless of how many edges are removed. Even with a single edge removal, the best result found is 0.175%, four times smaller in comparison to the best result found for the WS network with 1500 nodes.

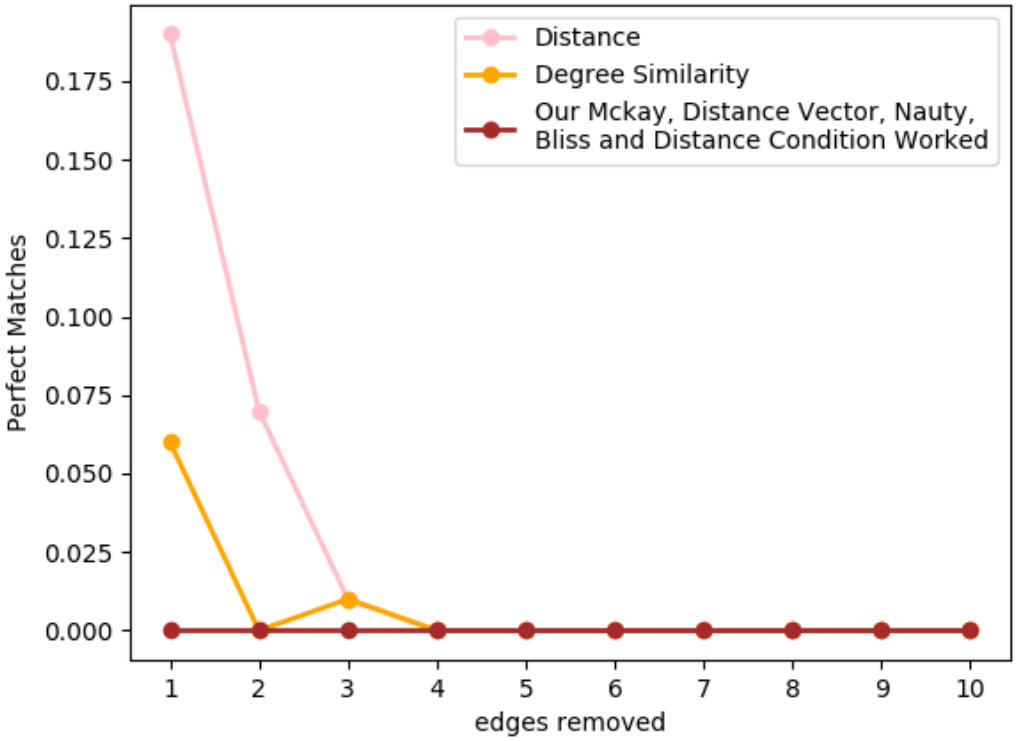


Figure 5.15: Perfect matches for $G(n = 100, p = 0.05)$ network.

Figures 5.16 and 5.17 show the results for the different edges mean metric. Note in Figure 5.16 that, for the different scenarios of edges removed, all the algorithms have similar behaviors as the number of edges removed increases and the distance algorithm performs better than all degree based and the distance vector algorithms. However, the ratio for the distance algorithm is never larger than 10% compared to at least one degree based algorithms. Interestingly, note in Figure 5.17 that even when the distance condition is not satisfied, on average, 50% of the edges are matched correctly when a single edge is removed.

A similar behavior is shown on the different nodes mean metric, presented in the Figures 5.18 and 5.19.

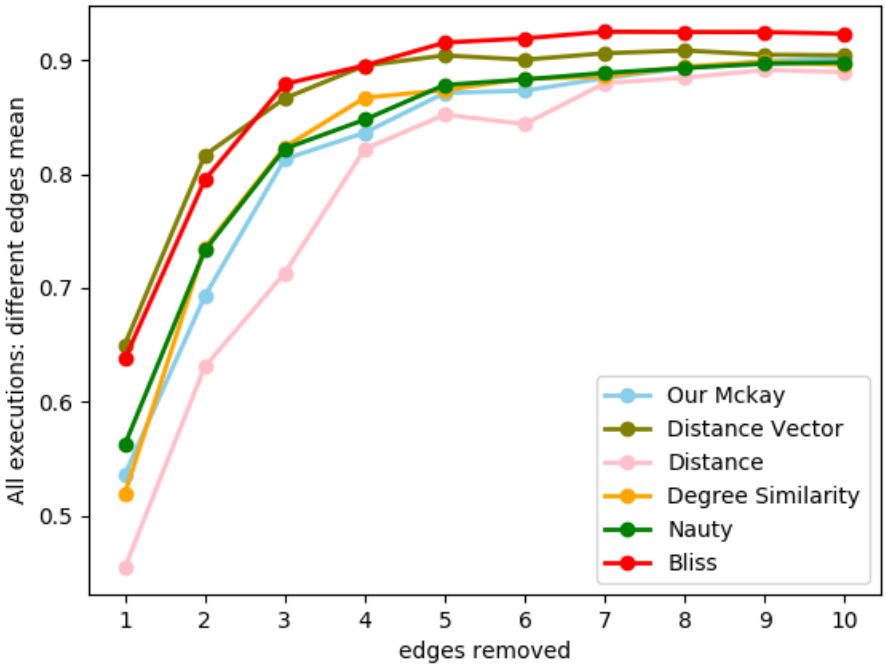


Figure 5.16: Different edges mean for $G(n = 100, p = 0.05)$ network.

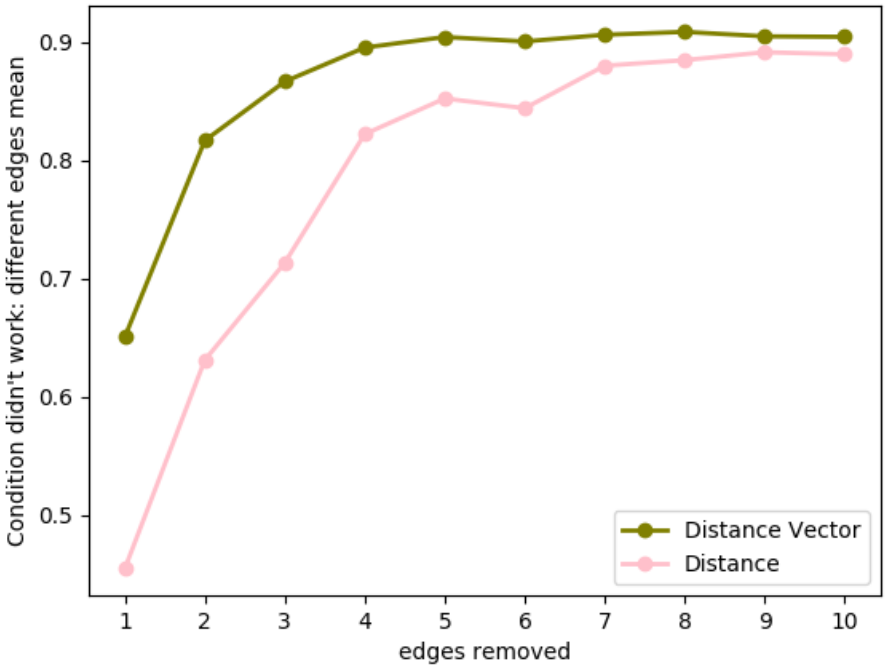


Figure 5.17: Distance condition not satisfied: different edges mean for $G(n = 100, p = 0.05)$ network.

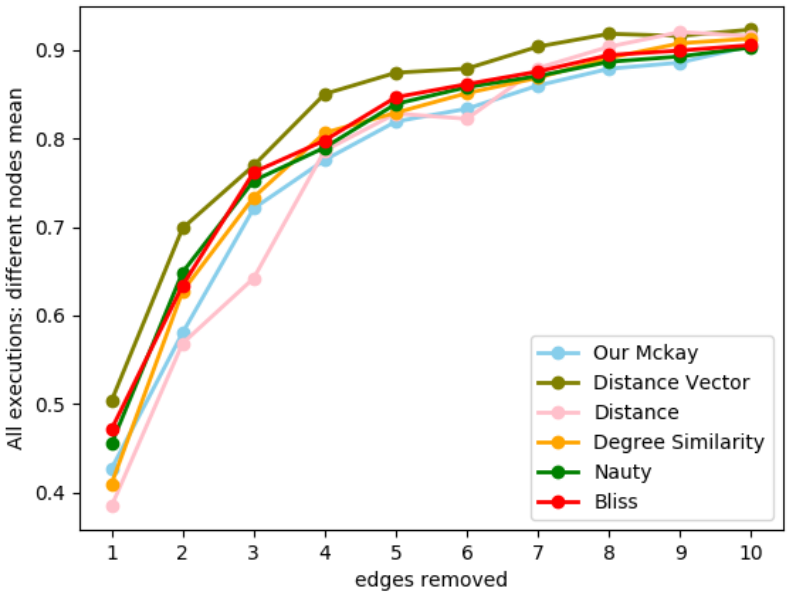


Figure 5.18: Different nodes mean for $G(n = 100, p = 0.05)$ network.

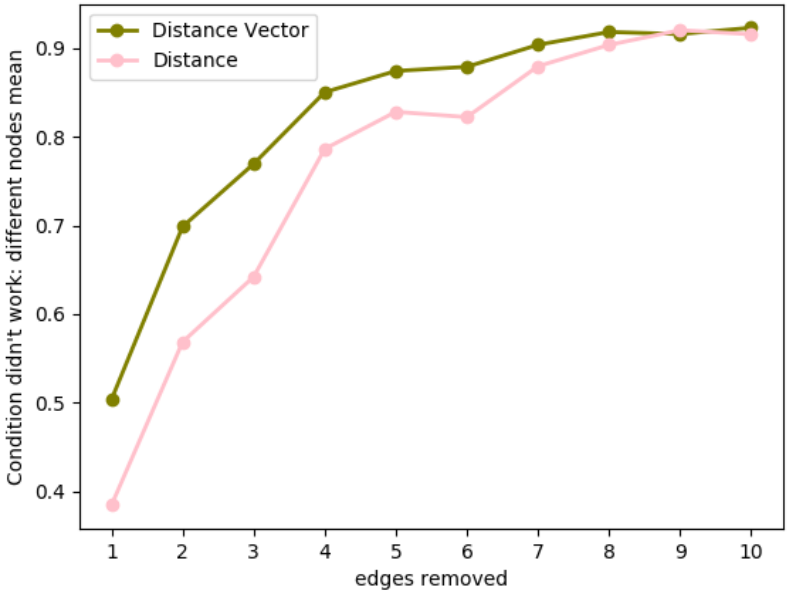


Figure 5.19: Distance condition not satisfied: different nodes mean for $G(n = 100, p = 0.05)$ network.

While the best results come from the distance algorithm, after just four edges are removed from the graph, all algorithms start having difficulty understanding that the two input graphs are similar.

Robustness Metrics

Table 5.3 shows the highest degree nodes metric for the $G(n = 100, p = 0.05)$ graphs generated. Note that the results are really small, which was expected. This model has an expected degree for each node and, independently of the node, they are all the same and equal to $np = 5$. Thus, the model generates networks with nodes that have similar degrees of approximately 5 and, hence, it is unlikely that nodes will have distinctive large degrees. This is one of the reasons why degree based algorithms do not show good performance.

Edges Removed	Mean	Standard Deviation
1	0.009	0.008
2	0.010	0.009
3	0.010	0.010
4	0.011	0.010
5	0.010	0.009
6	0.009	0.009
7	0.010	0.010
8	0.010	0.010
9	0.010	0.009
10	0.011	0.011

Table 5.3: Highest degree nodes values for $G(n = 100, p = 0.05)$ network.

Figure 5.20 shows the eccentricity metric for each number of edges removed. As the number of removed edges increases, the eccentricity tends to also increase. Note that when a single edge is removed, this metric is approximately 0.02 and when ten edges are removed, this metric goes to approximately 0.12, six times larger. Interestingly, the result is similar to the one presented for the WS model with 100 nodes.

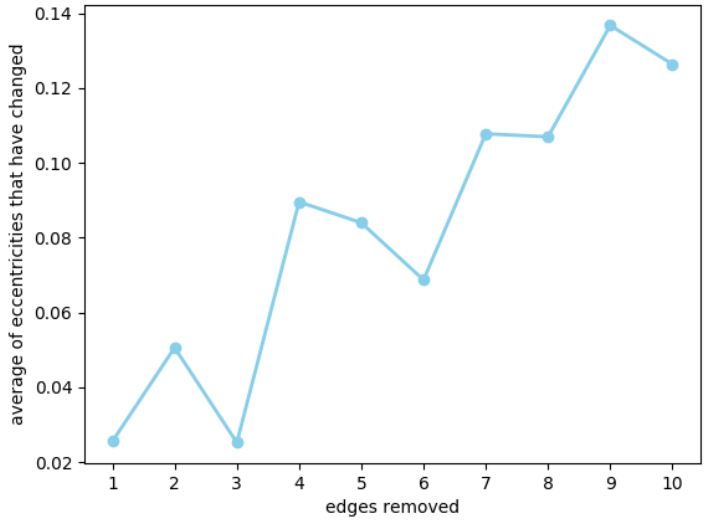


Figure 5.20: Eccentricity changed for $G(n = 100, p = 0.05)$ network.

Figure 5.21 shows the k units distance increase for each number of edges removed. The distances increase up to 6 units and some become infinite (G_2 becomes disconnected). Interestingly, even in a sparse random graph, there is some redundancy on the distances: when removing a single edge, less than 1% of node pairs have their distances increased and, even when 10 edges are removed, only 6% of node pairs have their distances changed. This is one of the reasons why the distance algorithms outperform the others, as shown in the evaluation metrics.

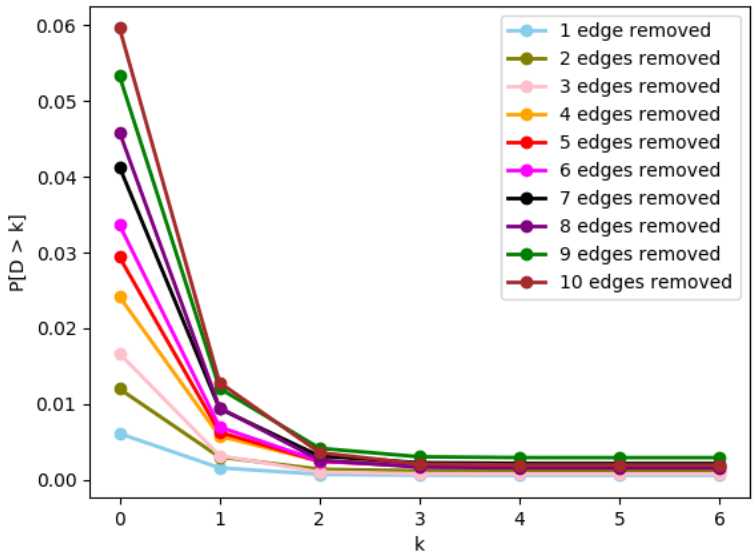


Figure 5.21: k units distance change for $G(n = 100, p = 0.05)$ network.

5.4 Results on Real Networks

While the previous section considered synthetic graphs, the six canonical labeling algorithms are now evaluated on the following real networks: the famous Zachary's Karate Club network [1], a given subgraph from Facebook consisting of circles (which is referenced as Facebook 686) and a collaboration network among professors from PESC[1]. Table 5.4 presents the main characteristics of each network and the following sections give more details about them, alongside their evaluation results.

	Karate Club	Facebook 686	PESC Collaboration
Number of Nodes	34	168	35
Number of Edges	78	1656	68
Average Degree	4.588	19.714	3.886
Maximum Degree	17	77	10
Minimum Degree	1	1	1
Average Distance	2.270	2.396	2.889
Maximum Distance	5	6	7
Minimum Distance	1	1	1

Table 5.4: Characteristics of the real networks evaluated.

5.4.1 Zachary's Karate Club [1]

Zachary's Karate Club is a social network of a university karate club that became famous in 2002 after it was used by Michelle Girvan and Mark Newman in their well-cited paper for community detection [40]. The network is an undirected graph representing 34 karate club members and 78 relationships between pairs of members who interacted outside the club. The data set is publicly available online.

The numbers of edges removed for this tests are $e = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ and, for each value of e, a hundred networks were generated, taking the initial graph and randomly removing some edges. In total, there are 1000 graph generated after removing edges, plus the original graph. The degree similarity algorithm had parameters set to $\delta_d = 1$ and $p = 0.3$.

Evaluation Metrics

The perfect match metric is presented in Figure 5.22. Note that the perfect match ratio decreases monotonically with the number of edges removed for all algorithms, dropping to zero when 6 edges are removed. This happens because the graph has

[1]PESC is the System Engineering and Computer Science graduate program offered by the Federal University of Rio de Janeiro.

a small number of edges and, after removing approximately 10% of the edges, the algorithms tend to fail. Moreover, the distance algorithms perform better than the degree algorithms, while the distance vector algorithm overlaps its results with the distance condition being satisfied and the distance algorithm outperforms the condition. Note that with a single edge removal, the perfect match ratio is twice as large for the distance algorithm in comparison to the distance condition being satisfied and four times larger for the distance algorithm in comparison to the degree similarity, that has the best performance between the degree based algorithms. The bliss and nauty algorithms had the same result for all edge removed values and overlapped on the plot, while after 3 or more edges are removed all algorithms, except the distance one, overlap at zero.

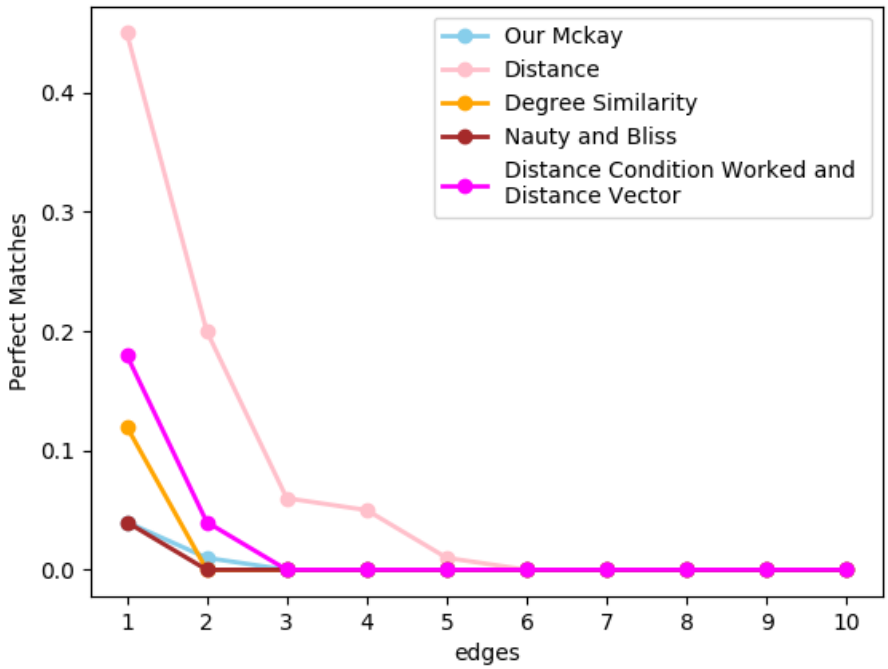


Figure 5.22: Perfect matches for Karate's network.

Figures 5.23 and 5.24 show the results for the different edges mean metric, for all the executions and for the executions where the distance condition was not satisfied, respectively. Figure 5.23 exhibits our implementation of the McKay canonical labeling algorithm usually outperforming the other five algorithms until 5 edges are removed, when the distance vector algorithm stars outperforming all others. Note that when removing 10 edges, all algorithms result in a similar different edges metric, around 60%.

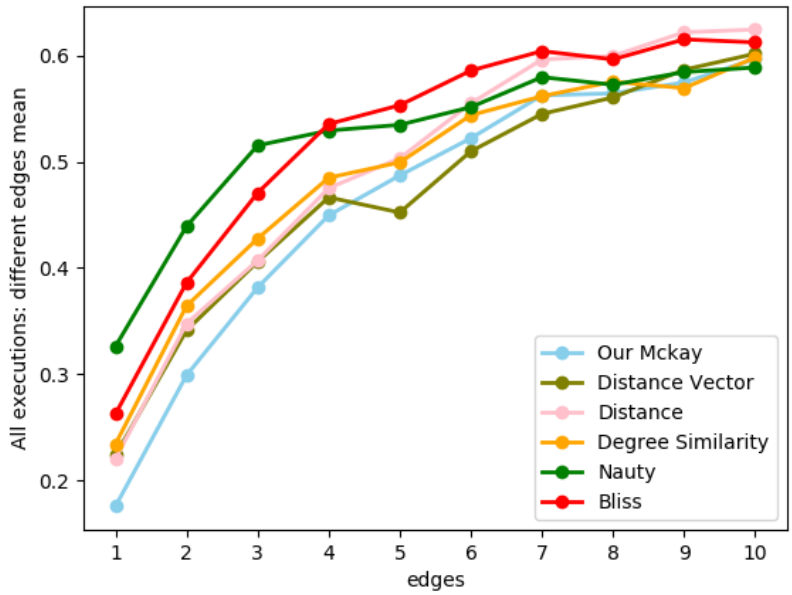


Figure 5.23: Different edges mean for Karate's network.

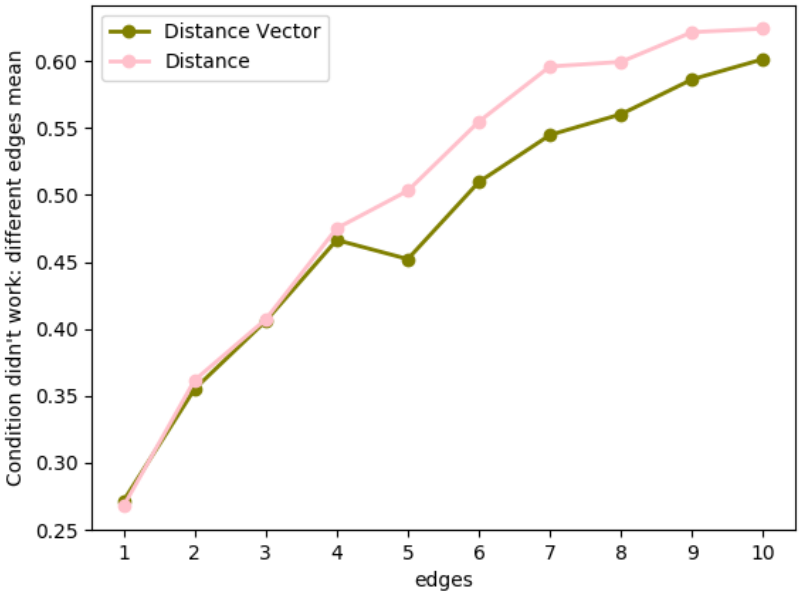


Figure 5.24: Distance condition not satisfied: different edges mean for Karate's network.

Figures 5.25 and 5.26 present the results for the different nodes mean metric, for all the executions and for the executions where the distance condition was not satisfied, respectively. Note that the distance based algorithms outperform all other algorithms. Interestingly, the distance based algorithms were superior in this node metric for any number of removed edges, while it only starts being superior in

the edge metric for more than 5 edges removed. This indicates that the distance algorithms are correctly matching more nodes with small degrees in comparison to our implementation of McKay, that is correctly matching fewer nodes, but with larger degrees.

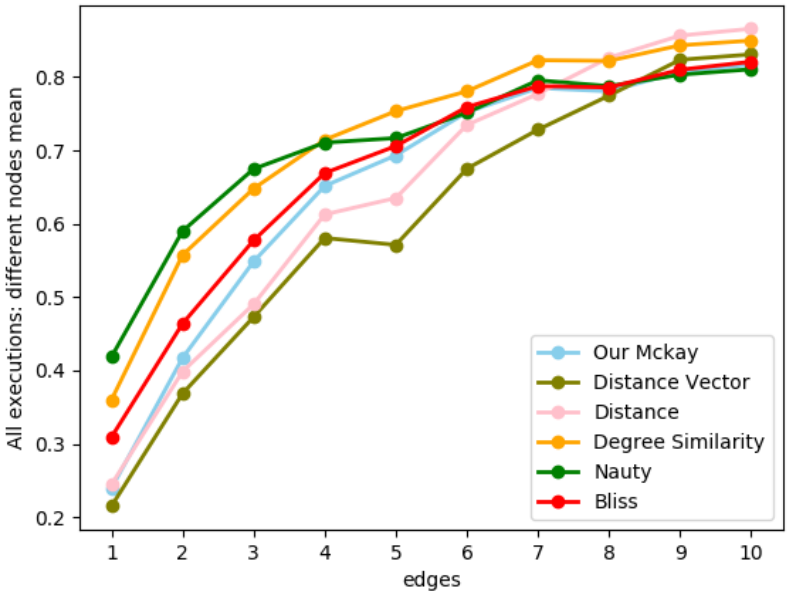


Figure 5.25: Different nodes mean for Karate's network.

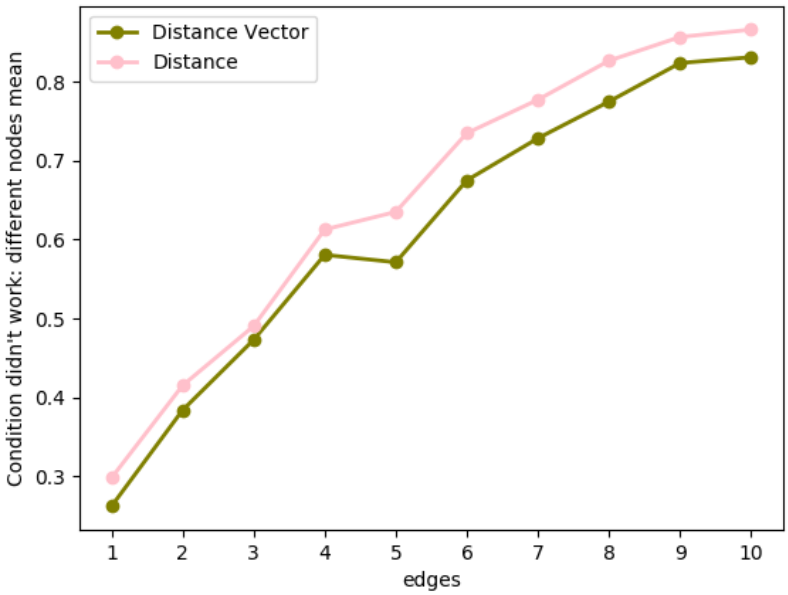


Figure 5.26: Distance condition not satisfied: different nodes mean for Karate's network.

Robustness Metrics

Table 5.5 shows the mean and standard deviation for the highest degree nodes metric for each number of edges removed from the graph. As shown, the mean is the same for each pair of graphs compared since the standard deviation is 0. Also, increasing the number of edges removed does not have an impact on the metric. The result is always 14.7%, which represents 5 nodes of the graph. This metric provides a lower limit to the different nodes mean metric for the degree based algorithms and it can be verified in Figure 5.25 that these algorithms always correctly match at least 20% of the nodes.

Edges Removed	Mean	Standard Deviation
1	0.147	0.0
2	0.147	0.0
3	0.147	0.0
4	0.147	0.0
5	0.147	0.0
6	0.147	0.0
7	0.147	0.0
8	0.147	0.0
9	0.147	0.0
10	0.147	0.0

Table 5.5: Highest degree nodes values for Karate's network.

Figure 5.27 shows the eccentricity metric mean value for different number of edges removed. It is clear that as the number of edges being removed from the graph increases, the metric also increases monotonically, to the point that after removing 10 edges, approximately 50% of the nodes had their eccentricity changed, which is approximately ten times larger in comparison to the result of a single edge removal. This is one of the reasons why the distance algorithm starts having worst performance as the number of edges removed increases.

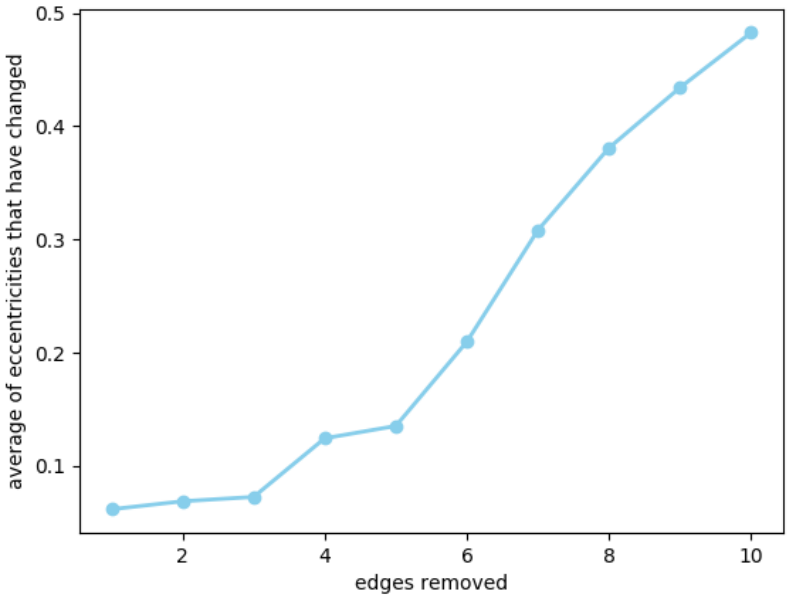


Figure 5.27: Eccentricity changed for Karate's network.

Figure 5.28 shows the k units distance increase for different number of edges removed. The distances increase up to 5 and, even after 10 edges are removed, approximately 85% of the distances remain the same. When removing just one edge, the amount of distances remaining the same is more than 98%, which is positive for the distance based canonical labeling algorithms.

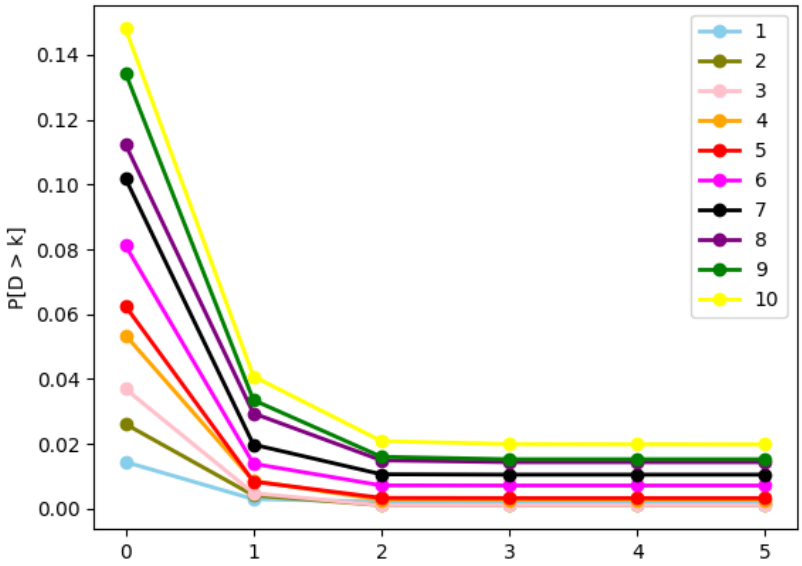


Figure 5.28: k units distance change for Karate's network.

5.4.2 Facebook Social Circle

This real network is available on the SNAP datasets [41] and belongs to a dataset that consists of circles from Facebook. The data is anonymized and was collected from survey participants using the Facebook app. The dataset has ten different networks and one of them was randomly chosen for this evaluation, named as Facebook 686 on the dataset, that has 168 nodes and 1656 edges. This is an undirected connected network, with a high average clustering coefficient and a great number of triangles.

The number of edges removed vary between 1 and 20, with $e = [1, 2, \ldots, 19, 20]$ and, for each value of e, a hundred networks were generated, taking the initial graph and randomly removing the edges. In total, there are 2000 graphs generated after removing some edges plus the original graph. The degree similarity algorithm had parameters set to $\delta_d = 1$ and $p = 0.3$.

Evaluation Metrics

Figure 5.29 shows the perfect match, where the distance algorithms outperform all other algorithms, with the distance algorithm also outperforming the distance condition being satisfied. With a single edge removal, the perfect match ratio for the distance algorithm is around 0.75, twice as large than the distance condition being satisfied and 0.2 larger than the second-best result, presented by the distance vector algorithm. With three removed edges, only the distance based algorithms are capable of finding a perfect match, where the distance vector perfectly matches the graphs approximately 18% of the times and the distance algorithm approximately 42% of the times. Interestingly, the distance algorithm is able of finding a perfect match even after 17 edges are removed, while there is no algorithm in the two random graph models analyzed so far that was capable of finding a perfect match after 10 edges are removed.

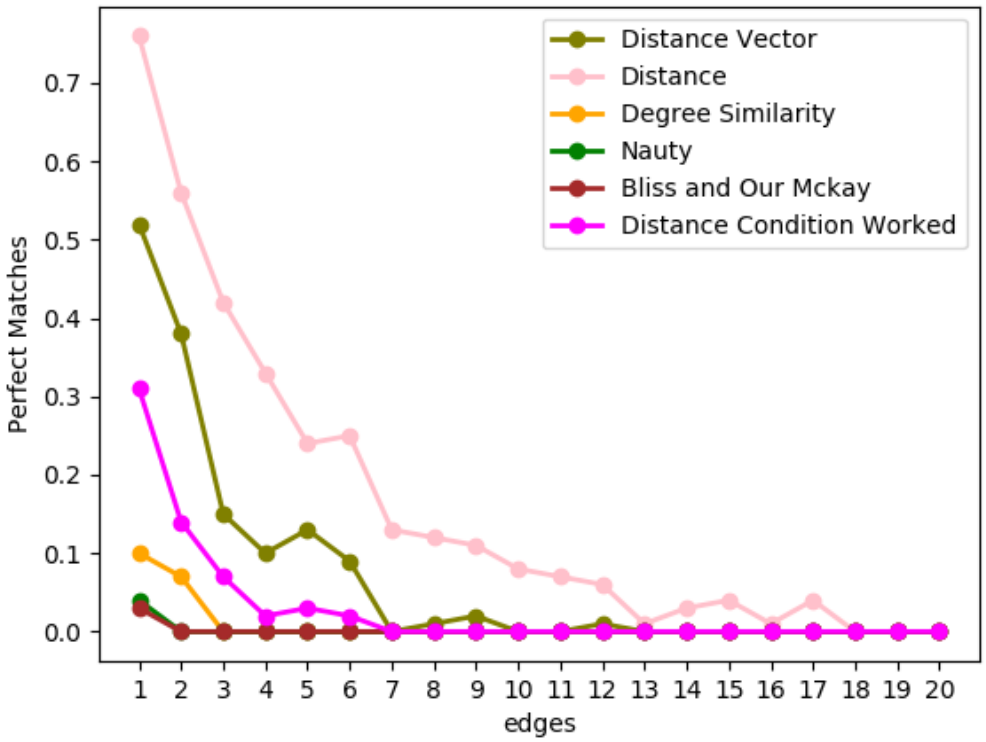


Figure 5.29: Perfect matches for Facebook's 686 network.

Figures 5.30 and 5.31 show the results for the different edges mean metric, where the distance algorithms perform better than all degree based algorithms. Our implementation of McKay's algorithm has close performance to the distance algorithms but is still always inferior to at least one of them. Nauty and degree similarity exhibit very similar results and, for a single edge removal, are twice as large in comparison to the distance based algorithms. As the number of edges removed increased, the distance between the results for these algorithms decreases, with the distance algorithms still performing better, but by a smaller factor. The Bliss algorithm is the one with the worst performance, where with a single edge removal it is four times larger in comparison to the distance algorithms and our implementation of McKay and twice as large in comparison to nauty and degree similarity.

The same behavior can be observed on the different nodes mean metric, presented in Figures 5.32 and 5.33.

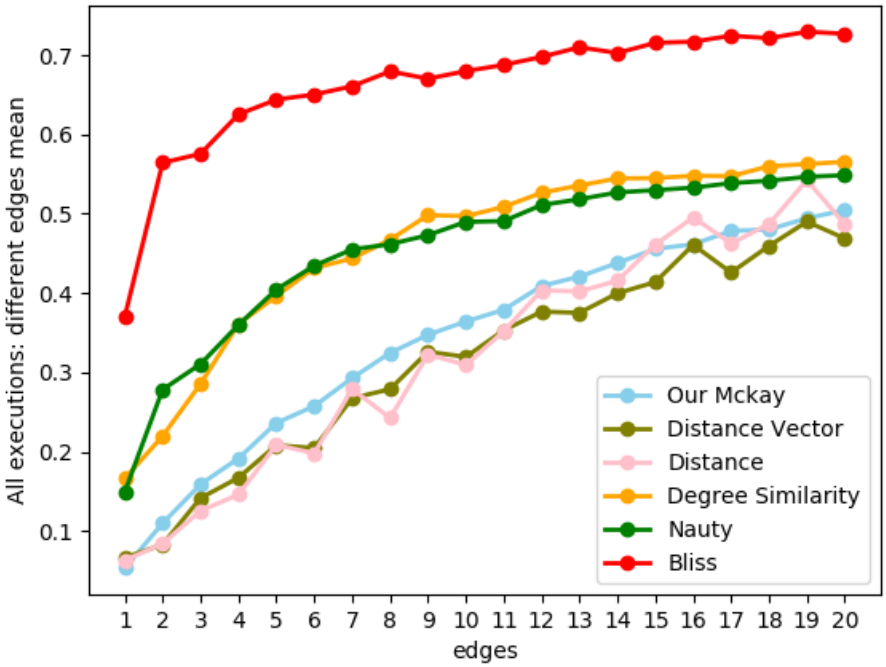


Figure 5.30: Different edges mean for Facebook's 686 network.

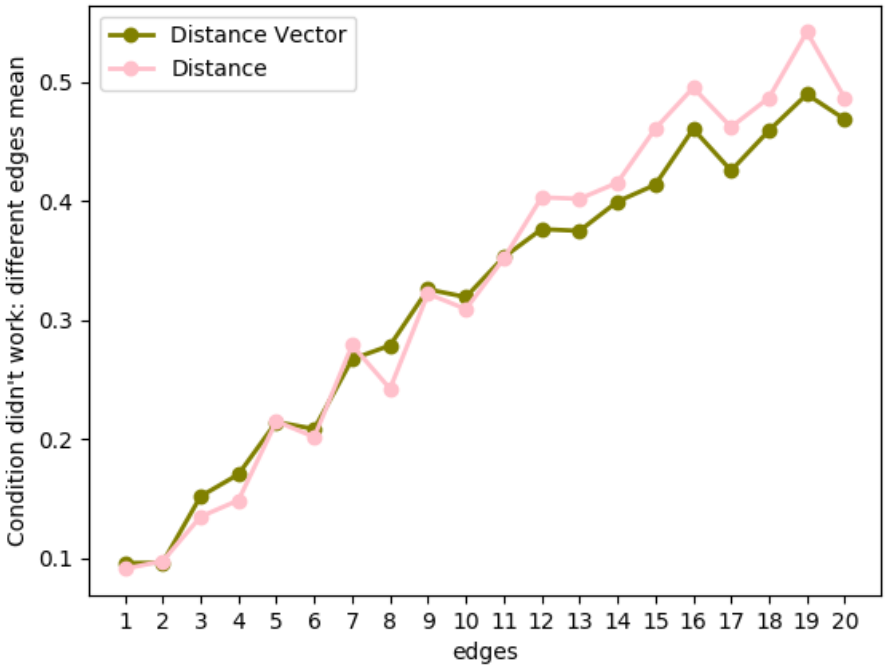


Figure 5.31: Distance condition not satisfied: different edges mean for Facebook's 686 network.

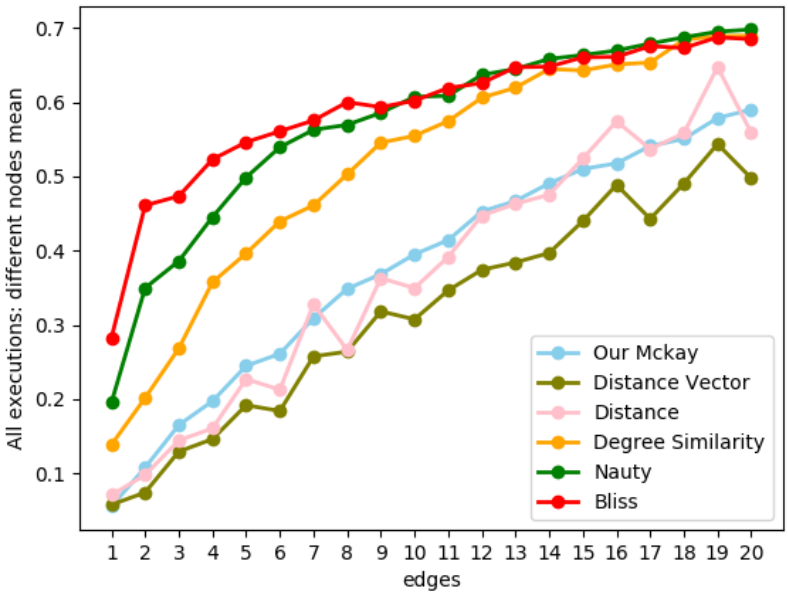


Figure 5.32: Different nodes mean for Facebook's 686 network.

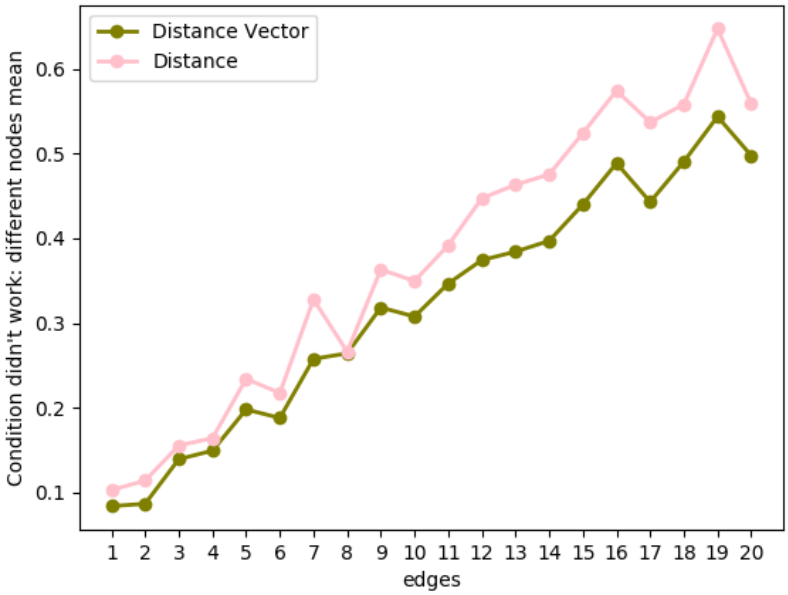


Figure 5.33: Distance condition not satisfied: different nodes mean for Facebook's 686 network.

Interestingly, the distance algorithm works better when finding a perfect match but the distance vector algorithm works better when aligning the networks, i.e., it aligns more edges correctly. This indicates that when the distance algorithm is not able to find a perfect match, its performance in aligning the networks is worst in comparison to the same scenario in the distance vector algorithm.

Robustness Metrics

Table 5.6 shows the mean and standard deviation for the highest degree nodes metric. The results for the different nodes mean show that all degree based algorithms always correctly match at least 30% of the nodes, which is higher than the 6% lower bound shown in the table.

Edges Removed	Mean	Standard Deviation
1	0.059	0.0
2	0.059	0.0
3	0.059	0.0
4	0.059	0.0
5	0.059	0.0
6	0.059	0.0
7	0.059	0.0
8	0.059	0.0
9	0.059	0.0
10	0.059	0.0
11	0.059	0.0
12	0.059	0.0
13	0.059	0.0
14	0.059	0.0
15	0.059	0.0
16	0.059	0.0
17	0.059	0.0
18	0.059	0.0
19	0.059	0.0
20	0.059	0.0

Table 5.6: Highest degree nodes values for Facebook's 686 network.

Figure 5.34 shows the eccentricity metric for each number of edges removed. Interestingly, the results are similar to the ones presented for the WS model.

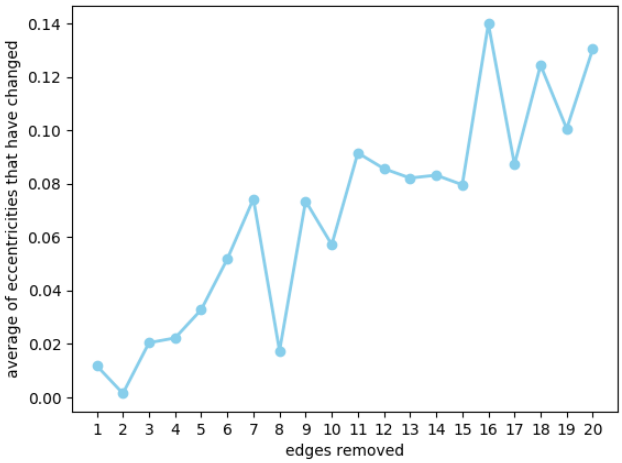


Figure 5.34: Eccentricity changed for Facebook's 686 network.

Figure 5.35 shows the k units distance increase for each amount of edges removed. The distances increase up to 3, with some distances becoming infinity. Note that even after removing 20 edges, only 0.7% of node pairs have their distances changed, and this number includes the pair of nodes directly affected by the edge removal (the distance between the nodes adjacent to an edge that is removed always change to a larger value). This behavior is positive for the distance based canonical labeling algorithms developed and one of the reasons they have better performance than all other algorithms. The results found for this network are smaller than the ones found for Karate's network, which is one of the reasons why it is easier to match edges and nodes in this network and to remove more edges without jeopardizing the performance.

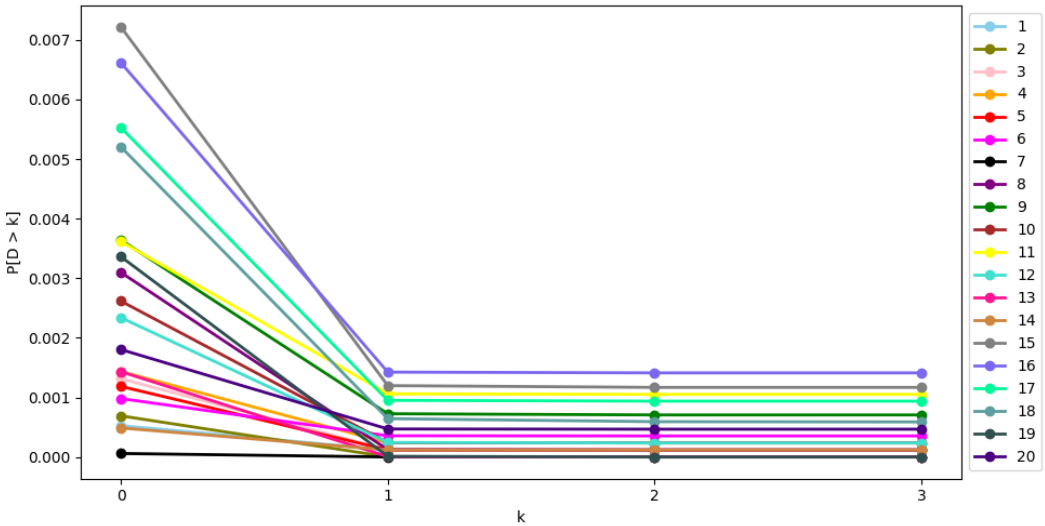


Figure 5.35: k units distance change for Facebook's 686 network.

This real network has redundancy on its distances and the distance based canonical labeling algorithms outperform all other algorithms on all metrics analyzed.

5.4.3 PESC's Collaboration Network

This real network was generated by the authors of this dissertation using public data from the DBLP website [2]. The nodes of the graph are 40 professors from PESC and the edges indicate collaboration between the professors, in this case, the publication of at least one paper together.

In order to generate the network, the name variations of each professor in the list was obtained from DBLP, since each person may have multiple ways of writing his name (for example, it is possible to have a paper authored by "Daniel Ratton Figueiredo" and another one by "Daniel R. Figueiredo"). The first step is normalizing the name of the 40 professors. Then, a crawler was developed to download and parse each professor's profile in DBLP and extract all the publications. The parser analyzed each publication and checked if amongst the authors were other professors from PESC, based on the name variations of all the 40 professors. If the parser finds two authors from PESC on the same publication, it creates an edge between them (if an edge already exists, nothing needs to happen).

At the time, there were 5 professors that did not haves publications with other professors from PESC in DBLP, so the final network ended up with 35 nodes since the idea was to create a connected graph.

The numbers of edges removed for this tests are $e = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ and, for each value of e, a hundred networks were generated, taking the initial graph and randomly removing edges. In total, there are 1000 graphs generated after edge removal. The degree similarity algorithm had parameters set to $\delta_d = 1$ and $p = 0.3$.

Evaluation Metrics

The perfect match is presented in Figure 5.36, showing a similar trend to the Karate network, where the distance based algorithms are superior but their performance deteriorates after 4 edges are removed. With a single edge removal, the best performance is given by the distance algorithm, which is more than three times larger in comparison to the distance condition being satisfied.

[2]The dblp computer science bibliography is the online reference for bibliographic information on major computer science publications.

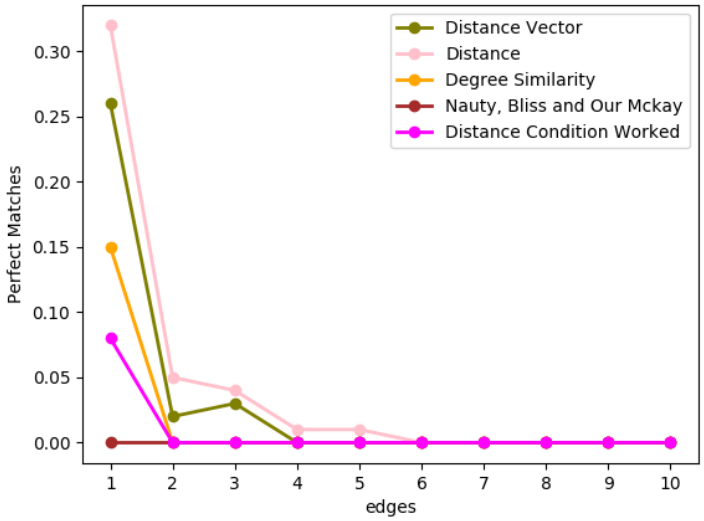


Figure 5.36: Perfect matches for PESC's collaboration network.

Figures 5.37 and 5.38 show the results for the different edges mean metric, where it is possible to note that, for all number of edges removed tested, the distance algorithms perform better than all degree based algorithms. In particular, the distance vector algorithm has the best performance for all number of edges removed, while the distance algorithm outperforms the distance vector algorithm in the perfect match metric until 5 edges are removed, when both algorithms are unable to find a perfect match. This indicates that when there is not a perfect match, the distance vector aligns the two networks better than the distance algorithm, as shown in Figure 5.38.

Figures 5.39 and 5.40 show the results for the different nodes metric, that behaves similarly to the edges metric.

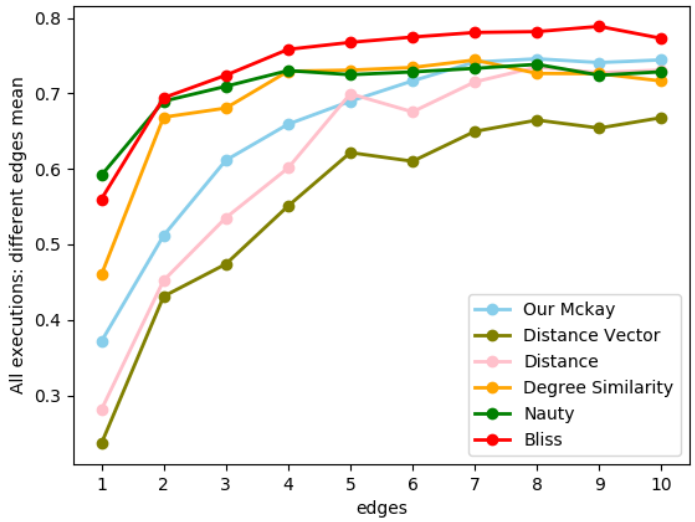


Figure 5.37: Different edges mean for PESC's collaboration network.

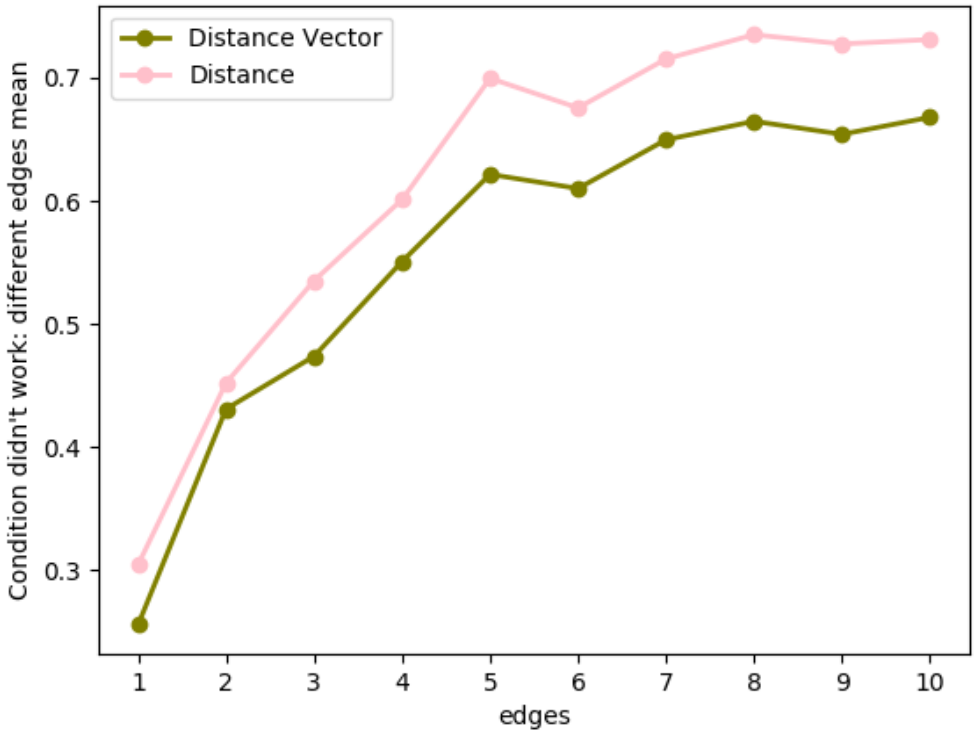


Figure 5.38: Distance condition not satisfied: different edges mean for PESC's collaboration network.

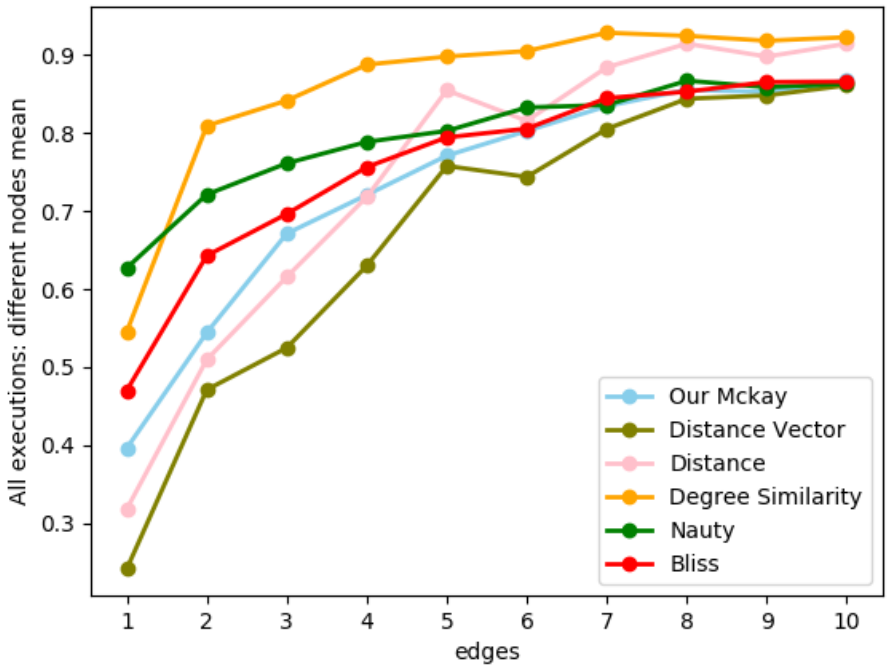


Figure 5.39: Different nodes mean for PESC's collaboration network.

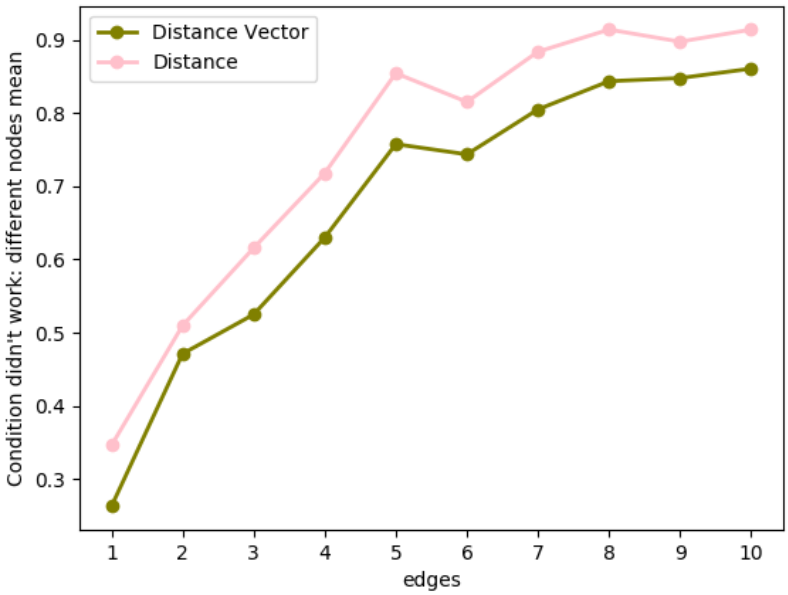


Figure 5.40: Distance condition not satisfied: different nodes mean for PESC's collaboration network.

Robustness Metrics

Table 5.7 shows the mean and standard deviation. The results are very similar to the ones from Facebook 686 network, sustaining the poor performance of degree based algorithms.

Edges Removed	Mean	Standard Deviation
1	0.057	0.0
2	0.057	0.0
3	0.057	0.0
4	0.057	0.0
5	0.057	0.0
6	0.057	0.0
7	0.057	0.0
8	0.057	0.0
9	0.057	0.0
10	0.057	0.0

Table 5.7: Highest degree nodes values for PESC's collaboration network.

Figure 5.41 shows the eccentricity metric for different number of edges removed. The result is similar to the Karate network, with the exception that eccentricity changes even faster in this network.

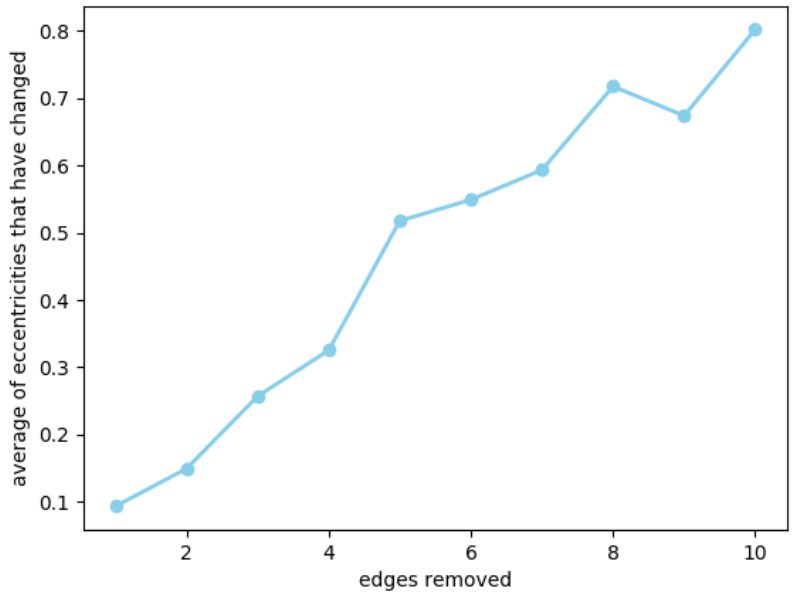


Figure 5.41: Eccentricity changed for PESC's collaboration network.

Figure 5.42 shows the k units distance increase for different number of edges removed. It also behaves similarly to the Karate network as the number of removed edges increase, with the difference that this network is more sensitive to edge removal, hence, having more distances changed. For ten removed edges, more than 25% of the distances change, which is explained by the fact that the network is really sparse and ten edges represent almost 15% of the total edges.

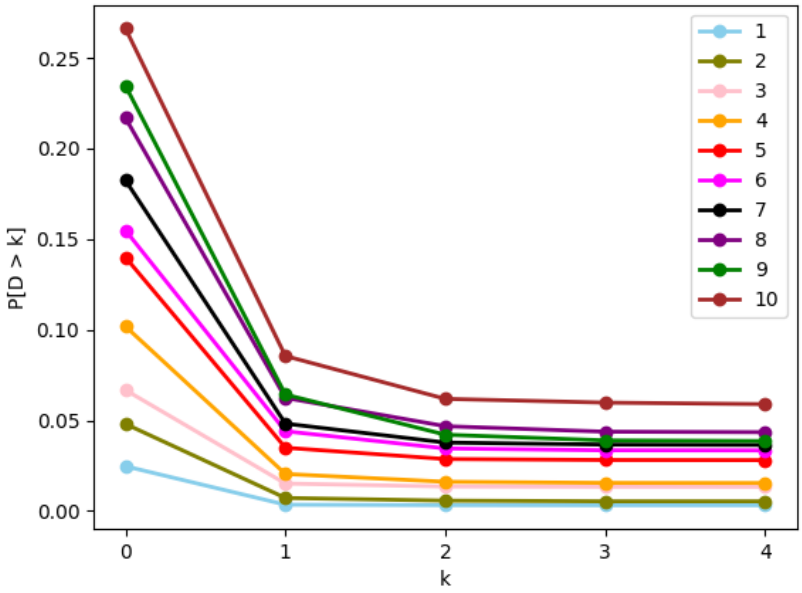


Figure 5.42: k units distance change for PESC's collaboration network.

Chapter 6

Conclusion and Future Work

This dissertation tackles the graph matching problem with canonical labeling algorithms. In particular, it proposes a novel distance-based approach that can solve the graph matching problem under some condition while always solving graph isomorphism. Two algorithms are implemented based on distances, called distance and distance vector, that only differ in the refinement procedure. The first one splits the nodes based on the minimum distance between a node and a cell, while the latter splits the nodes based on the distances between a node and every node from a cell, creating a vector filled with all these distances. While the distance algorithm uses less information and is more likely to succeed, the distance vector algorithm is faster since it can break more cells in each refinement.

A proof of the algorithms correctness for the isomorphism problem and the graph matching problem under a certain condition are provided, where this condition states that the algorithms will always match the nodes correctly from two graphs with the same distance matrices except for some values that are 1 in one of them and 2 on the other.

A variation of one state of the art degree based canonical labeling algorithm was also proposed and implemented, called degree similarity. The idea is to understand if it is possible to still use degrees to drive the canonical labeling algorithm and solve the graph matching problem on similar graphs.

A simple edge removal model was created to generate similar, but not isomorphic graphs. It has two parameters, a graph and a number of edges to remove, and returns a new graph that is a copy of the input graph missing some edges, that were removed uniformly at random. This pair of similar graphs is then passed through each canonical labeling algorithm. An extensive evaluation on random graph models and real networks under this simple edge removal model show that, in general, the distance based algorithms are the only ones that can find a perfect match between similar graphs. Results also show that the real networks evaluated do have path redundancy and, hence, the distances are robust to a small number of edge removals,

making the proposed algorithms outperform the others in all metrics investigated.

However, it is important to say that redundancy in distances is necessary for the proposed algorithms and, when the number of removed edges increases, this redundancy decreases and the performance of the algorithm in matching the graphs decreases. This is one of the reasons why the algorithms tend to have good performance when only a small number of edges are removed.

6.1 Future Work

As part of future work, the algorithms should be evaluated on more networks and other models for creating similar graphs to strengthen the conclusions. It also would be interesting to test the algorithms on larger networks, with more than 1500 nodes, which is the maximum number of nodes evaluated in this work. However, since this algorithm does not run in polynomial time, as the number of nodes increases so does the time to run the algorithm. Thus, improving its running time is also a line for future work.

Conditions for a Perfect Match

This work presented one structural condition where the distance algorithms always solve the graph matching problem. However, the results indicate that this condition is sufficient, but not necessary since the algorithm can still solve the graph matching problem when the condition is not satisfied. It would be interesting to understand these other scenarios where the algorithms work and provide theoretical proofs of their correctness based on these conditions.

Refinements

This work presented two distance and two degree refinement procedures to drive the canonical labeling algorithms. As seen, the distance based algorithms work better than any degree based algorithm on most of the networks, especially on real networks where redundancy is present. However, when the number of removed edges increases, the performance of all algorithms decreases fast. Thus, it would be interesting to design other refinement procedures that are more robust to edge removal and have better performance when the number of edges removed from the original graph is larger.

Evaluation on Other Similar Graphs

This work proposed the analysis of canonical labeling algorithms on two similar but not isomorphic graphs, G_1 and G_2, with G_2 being a subgraph of G_1, created after

randomly removing k edges from G_1. It would also be interesting to study other similar graphs, especially where G_2 is not a subgraph of G_1. To tackle this problem, there is a need to create a new model to generate similar graphs, like receiving a graph and randomly removing and adding a small number of edges to it, creating a new graph. This is interesting mostly because it would expand the functionality of the developed algorithms, making it possible to be used on other common graph matching problems.

References

[1] ZACHARY, W. W. "An Information Flow Model for Conflict and Fission in Small Groups", *Journal of Anthropological Research*, v. 33, n. 4, pp. 452–473, 1977.

[2] WATTS, D. J., STROGATZ, S. H. "Collective dynamics of 'small-world' networks", *Nature*, v. 393, n. 6684, pp. 440–442, 1998.

[3] SCHÖNING, U. "Graph Isomorphism is in the Low Hierarchy", *Journal of Computer and System Sciences*, v. 37, n. 3, pp. 312–323, 1988.

[4] BABAI, L., LUKS, E. M. "Canonical Labeling of Graphs". In: *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pp. 171–183, 1983.

[5] MCKAY, B. D. "Backtrack programming and the graph isomorphism problem", *M. Sc. Thesis, University of Melbourne*, 1976.

[6] MCKAY, B. D. "Practical Graph Isomorphism", *Congressus Numerantium*, v. 30, pp. 45–87, 1980.

[7] MCKAY, B. D., PIPERNO, A. "Practical graph isomorphism, II", *Journal of Symbolic Computation*, v. 60, n. 0, pp. 94–112, 2014.

[8] JUNTTILA, T. A., KASKI, P. "Engineering an Efficient Canonical Labeling Tool for Large and Sparse Graphs". In: *Proceedings of the Nine Workshop on Algorithm Engineering and Experiments, ALENEX 2007*, 2007.

[9] BABAI, L., KUCERA, L. "Canonical Labelling of Graphs in Linear Average Time". In: *20th Annual Symposium on Foundations of Computer Science*, pp. 39–46, 1979.

[10] MASON, O., VERWOERD, M. "Graph theory and networks in Biology", *IET Systems Biology*, v. 1, n. 2, pp. 89–119, 2007.

[11] PRŽULJ, N., CORNEIL, D. G., JURISICA, I. "Efficient estimation of graphlet frequency distributions in protein–protein interaction networks", *Bioinformatics*, v. 22, n. 8, pp. 974–980, 2006.

[12] BENNETT, J., LANNING, S. "The Netflix Prize". In: *Proceedings of the KDD Cup Workshop 2007*, pp. 3–6. Association for Computing Machinery (ACM), 2007.

[13] TABAK, P., FIGUEIREDO, D. R. "Algoritmo Eficiente para Quebra de Privacidade em Redes Sociais Anonimizadas". In: *Anais do III Encontro de Teoria da Computação*. Sociedade Brasileira de Computação (SBC), 2018.

[14] BACKSTROM, L., DWORK, C., KLEINBERG, J. M. "Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography". In: *Proceedings of the 16th International Conference on World Wide Web, WWW 2007*, pp. 181–190, 2007.

[15] HAY, M., MIKLAU, G., JENSEN, D. D., et al. "Resisting structural re-identification in anonymized social networks", *Proceedings of the Very Large Databases Endowment (PVLDB)*, v. 1, n. 1, pp. 102–114, 2008.

[16] RIESEN, K., JIANG, X., BUNKE, H. "Exact and Inexact Graph Matching: Methodology and Applications". In: *Managing and Mining Graph Data*, pp. 217–247, 2010.

[17] CONTE, D., FOGGIA, P., SANSONE, C., et al. "Thirty Years Of Graph Matching In Pattern Recognition", *International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI)*, v. 18, n. 3, pp. 265–298, 2004.

[18] MITZENMACHER, M., MORGAN, T. "Reconciling Graphs and Sets of Sets", *Computing Research Repository (CoRR)*, v. abs/1707.05867, 2017.

[19] FRIEDLAND, S. "On the graph isomorphism problem", *Computing Research Repository (CoRR)*, v. abs/0801.0398, 2008.

[20] MCKAY, B. D. "Graph Isomorphism". In: *Encyclopedia of Algorithms*, pp. 875–879, 2016.

[21] PAHL, A. J., DAMRATH, R., PAHL, F. *Mathematical Foundations for Computational Engineering: A Handbook*. Springer-Verlag, 2001.

[22] HARTKE, S., RADCLIFFE, A. "McKay's Canonical Graph Labeling Algorithm", *Contemporary Mathematics book series*, v. 479, 2013.

[23] MIYAZAKI, T. "The complexity of McKay's canonical labeling algorithm". In: *Groups and Computation, Proceedings of a DIMACS Workshop*, pp. 239–256, 1995.

[24] MCKAY, B. D., PIPERNO, A. *nauty and Traces User's Guide*, 2016.

[25] PIPERNO, A. "Search Space Contraction in Canonical Labeling of Graphs (Preliminary Version)", *Computing Research Repository (CoRR)*, v. abs/0804.4881, 2008.

[26] DE OLIVEIRA OLIVEIRA, M., GREVE, F. "A new refinement procedure for graph isomorphism algorithms", *Electronic Notes in Discrete Mathematics*, v. 19, pp. 373–379, 2005.

[27] CODENOTTI, P. "Distinguishing vertices of inhomogeneous random graphs". 2013.

[28] SHERVASHIDZE, N., SCHWEITZER, P., VAN LEEUWEN, E. J., et al. "Weisfeiler-Lehman Graph Kernels", *Journal of Machine Learning Research*, v. 12, pp. 2539–2561, 2011.

[29] CAI, J., FÜRER, M., IMMERMAN, N. "An optimal lower bound on the number of variables for graph identifications", *Combinatorica*, v. 12, n. 4, pp. 389–410, 1992.

[30] BOLLOBÁS, B. "Distinguishing Vertices of Random Graphs". 1982.

[31] HOPCROFT, J. E., KARP, R. M. "An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs", *SIAM J. Comput.*, v. 2, n. 4, pp. 225–231, 1973.

[32] ERDÖS, P., RÉNYI, A. "On Random Graphs I", *Publicationes Mathematicae Debrecen*, v. 6, pp. 290–297, 1959.

[33] NARAYANAN, A., SHMATIKOV, V. "De-anonymizing Social Networks". In: *30th IEEE Symposium on Security and Privacy*, pp. 173–187, 2009.

[34] PEDARSANI, P., GROSSGLAUSER, M. "On the privacy of anonymized networks". In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1235–1243, 2011.

[35] YARTSEVA, L., GROSSGLAUSER, M. "On the performance of percolation graph matching". In: *Conference on Online Social Networks*, pp. 119–130, 2013.

[36] VENKATESAN, R., VAZIRANI, V. V., SINHA, S. "A Graph Theoretic Approach to Software Watermarking". In: *Information Hiding*, pp. 157–168, 2001.

[37] COLLBERG, C. S., HUNTWORK, A., CARTER, E., et al. "More on graph theoretic software watermarks: Implementation, analysis, and attacks", *Inf. Softw. Technol.*, v. 51, n. 1, pp. 56–67, 2009.

[38] BENTO, L. M. S., BOCCARDO, D. R., MACHADO, R. C. S., et al. "Towards a Provably Resilient Scheme for Graph-Based Watermarking". In: *Graph-Theoretic Concepts in Computer Science*, pp. 50–63, 2013.

[39] BARABÁSI, A. *Linked - how everything is connected to everything else and what it means for business, science, and everyday life*. Plume, 2003.

[40] GIRVAN, M., NEWMAN, M. E. J. "Community structure in social and biological networks", *Proceedings of the National Academy of Sciences*, v. 99, n. 12, pp. 7821–7826, 2002.

[41] LESKOVEC, J., KREVL, A. "SNAP Datasets: Stanford Large Network Dataset Collection". 2014.