Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

COPPE UFRJ

A PERSONALIZED PAGERANK FINGERPRINT FRAMEWORK FOR SEEDLESS GRAPH MATCHING

Rafael Gonçalves Damasceno

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
Setembro de 2024

A PERSONALIZED PAGERANK FINGERPRINT FRAMEWORK FOR SEEDLESS GRAPH MATCHING

Rafael Gonçalves Damasceno

Orientador: Daniel Ratton Figueiredo

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2024

Damasceno, Rafael Gonçalves

A Personalized PageRank Fingerprint Framework for Seedless Graph Matching/Rafael Gonçalves Damasceno. – Rio de Janeiro: UFRJ/COPPE, 2024.

XII, 48 p.: il.; 29, 7cm.

Orientador: Daniel Ratton Figueiredo

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2024.

Referências Bibliográficas: p. 40 – 42.

1. Graph Matching. 2. Network Alignment. 3. Personalized PageRank. 4. PageRank. I. Figueiredo, Daniel Ratton. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

Agradeço aos meus pais, Gloria e Roberto, e à minha avó, Dalva. Sem o apoio incondicional esta jornada acadêmica não teria sido possível. Vocês sempre acreditaram em mim e me incentivaram a seguir meus sonhos.

À minha companheira, Mariana, meu sincero agradecimento. Sua compreensão e encorajamento foram fundamentais durante todo este processo. Obrigado por estar ao meu lado nos momentos de dificuldade e por celebrar comigo cada pequena vitória. Sua presença tornou esta caminhada muito mais leve e significativa.

Agradeço também à Universidade Federal do Rio de Janeiro, onde realizei minha graduação e este mestrado, por proporcionar um ambiente de excelência acadêmica. Esta instituição foi essencial para o meu desenvolvimento profissional e pessoal.

Aos amigos que fiz na universidade, agradeço imensamente por cada apoio e por todos os momentos de descontração.

Por fim, agradeço ao meu orientador, Daniel Ratton Figueiredo. Sua orientação, conhecimento e dedicação foram essenciais para a concretização desta dissertação.

A todos vocês, meu muito obrigado.

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

UMA FRAMEWORK COM REPRESENTAÇÕES BASEADA EM PERSONALIZED PAGERANK PARA EMPARELHAMENTO DE GRAFOS SEM SEMENTES

Rafael Gonçalves Damasceno

Setembro/2024

Orientador: Daniel Ratton Figueiredo

Programa: Engenharia de Sistemas e Computação

Grafos são usados para modelar problemas e codificar relacionamentos entre entidades em vários contextos, abrangendo estudos que vão desde redes sociais até interações biológicas. Um problema clássico em teoria dos grafos envolve encontrar o mapeamento de vértices entre dois grafos que exibem maior similaridade. Esse problema é conhecido como emparelhamento de grafos, também referido como alinhamento de redes. O emparelhamento de grafos depende fortemente da estrutura da rede e surge em contextos diversos devido à importância de identificar estruturas semelhantes em redes. Nesta dissertação, apresenta-se uma nova framework que não requer sementes, usa representações baseadas em Personalized PageRank, e utiliza um algoritmo em rodadas para mapear gradualmente os nós. Em cada rodada, o algoritmo define pares de âncoras que irão aprimorar as representações dos nós para, então, decidir os emparelhamentos na rodada subsequente. Este trabalho formaliza a definição de uma nova representação de vértices com base no Personalized PageRank, bem como uma métrica de similaridade para comparar vértices usando essa representação introduzida. Os resultados de avaliação demonstram valores de acurácia de até 97% em média, considerando emparelhamentos corretos em pares de grafos que foram submetidos a um processo de edge sampling com uma probabilidade de remoção de aresta de 20% no modelo Barabási–Albert.

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

A PERSONALIZED PAGERANK FINGERPRINT FRAMEWORK FOR SEEDLESS GRAPH MATCHING

Rafael Gonçalves Damasceno

September/2024

Advisor: Daniel Ratton Figueiredo

Department: Systems Engineering and Computer Science

Graphs are used to model problems and encode relationships between entities in various contexts, encompassing studies ranging from social networks to biological interactions. A classic problem in graph theory involves finding the node mapping between two graphs that exhibit high similarity. This problem is known as graph matching, also referred to as network alignment. Graph matching heavily relies on the structure of the network and arises in diverse contexts due to the importance of identifying similar structures in networks. In this dissertation, we introduce a novel seedless framework based on Personalized PageRank fingerprints, utilizing an algorithm in rounds to gradually map nodes. In each round, the algorithm defines anchor-mapped pairs to enhance fingerprints and then decide matchings in the subsequent round. This work formalizes the definition of a new node fingerprint based on Personalized PageRank, as well as a metric for comparing nodes using the introduced fingerprint. Our evaluation results demonstrate accuracy values of up to 97% on average, considering correct matchings across pairs of graphs that have been subjected to an edge sampling process with a 20% probability of edge removal in the Barabási–Albert model.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

A variety of problems in computer science are modeled using graphs to encode relationships between entities. This representation can be defined such that a graph G is given by $G = (V, E)$, where V is a set of vertices (also known as nodes) and vertices are connected by a set of edges E. Graph modeling proves to be a powerful tool that abstracts a problem, allowing a simple representation and making it possible to approach the problem with different algorithms known in graph theory.

At the same time, several themes in society and nature can be represented by networks. Thus, the field of network science proves to be highly interdisciplinary [1, 2], encompassing studies ranging from social networks to biological interactions. In each study, the meaning of vertices and edges may be different, in a social network vertices may represent users, and edges the existence of friendship, while in a protein-protein network, vertices may represent proteins, and edges indicate interaction between them. Within this context, a graph problem may address different topics depending on the specific network where it is being studied.

The graph matching problem, also known as network alignment, involves finding a node-to-node correspondence between two graphs. The resulting node mapping must be such that it maximizes the structural similarity between the graphs, and will associate the label of a vertex in one network with the label of a vertex in the other network. This problem heavily relies on the network's structure and arises in various contexts due to the importance of identifying similar structures. Typically, this problem involves two similar, yet distinct networks, where there is a significant correlation between the entities (vertices) of one network and those of the other. This correlation suggests that the mapping entities either perform the same function within their respective networks or represent the same entity. It is important to note that when this problem involves identifying corresponding vertices between two networks with the same number of nodes and edges, and we are interested in determining if the mapping preserves the adjacency structure, the problem is referred to as graph isomorphism problem.

Consider the context of social networks to illustrate the application of a graph matching problem. In two distinct networks, such as mutual friendships on a social media platform (e.g., Facebook or Instagram) and message exchanges on a messaging application (e.g., WhatsApp), it is expected that there will be a structural similarity despite the networks not being identical. For instance, users who are friends on Facebook are more likely to also communicate via WhatsApp. In this example, solving the graph matching problem could allow the association of user profiles on the social media platform with phone numbers found in the messaging network by identifying the node mapping between the networks that represent the same user.

As an illustrative example, we can examine Figure 1.1 as representing two subsets in different networks: one from a social media platform where edges denote mutual friendships, and another from a messaging application where edges denote message exchanges between users. These networks share notable similarities. For instance, in the social media network, node d acts as a hub, and similarly, node 6 plays the same role in the messaging application network, both with the highest number of connections. Therefore, without additional information, the vertices are likely to correspond to each other. At the same time, it's worth noting that some nodes within the same network are structurally equivalent, making them harder to match, such as nodes a or b in the social media network. Despite the networks not

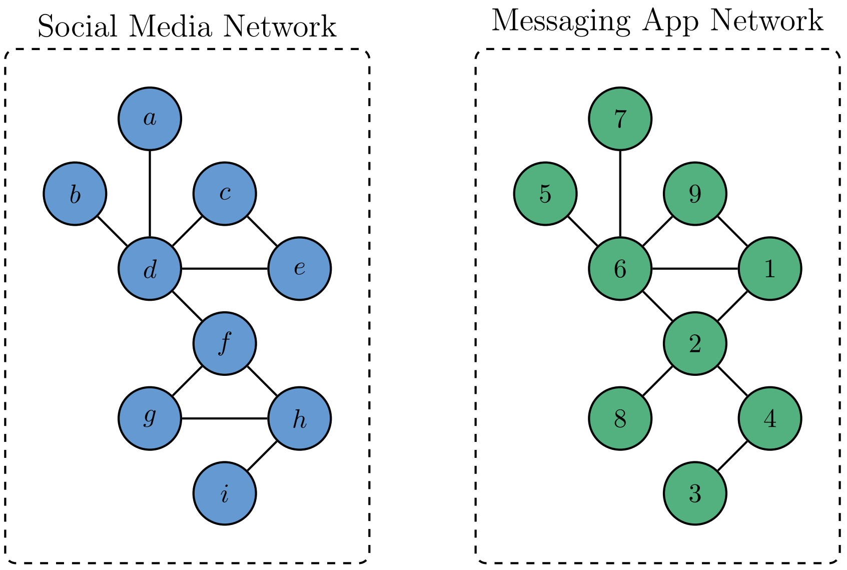


Figure 1.1: Comparison of a social media network subset (e.g., Facebook) and a messaging application network subset (e.g., WhatsApp). In the social media network, nodes represent users and edges indicate mutual friendships. In the messaging application network, nodes also represent users, but edges indicate message exchanges.

being identical, with edges present in one network but absent in the other, performing graph matching between these networks to identify corresponding nodes could potentially reveal the phone numbers associated with each user. This highlights privacy concerns, as the ability to cross-reference data from different platforms.

Literature has shown that graph matching between two social networks can lead to the de-anonymization of users in a previously anonymized network, considering only the structural similarity between an anonymized network and an auxiliary network [3]. The work conducted used a graph of users from Flickr and an anonymized graph of users from Twitter, two social networks, and was able to re-identify users in the anonymized Twitter graph with only a 12% error rate. This result can raise several discussions about privacy and information security.

In the field of biology, other works have demonstrated that the application of graph matching between two biomolecular interaction networks, such as protein–protein interaction networks (PPI), can identify common interaction patterns [4]. Solving this problem within this context allows for mapping proteins with similar roles between two different organisms based on their connectivity patterns. Similar studies are also applied to other types of biological networks, such as gene co-expression, metabolic, or gene regulatory networks.

To address the graph matching problem, different approaches aim to represent network nodes, two prominent techniques include the use of embeddings and fingerprints. Fingerprints refer to the structural characteristics of a vertex or a set of these characteristics, including for example degree, eccentricity, clustering coefficient, and the distance to a reference vertex. These attributes encode details about the role of each node within the network topology. On the other hand, embeddings result from transforming the graph structure into a low-dimensional space, where each vertex is represented by a single d-dimensional vector [5], known as the vertex embedding. This transformation facilitates the application of machine learning algorithms by enabling models to more effectively understand real-world data domains.

This scenario allows for exploring diverse methods of vertex representation, ensuring their roles within the network topology are effectively captured. Additionally, it enables the exploration of various techniques to calculate node similarity and determining the final matching solution for the problem. Several methodologies can be employed to address the graph matching problem, including Bayesian methods [6], percolation [7], machine learning [8], and optimization techniques.

1.2 Contributions

The work described in this dissertation addresses the problem of graph matching by proposing a novel framework based on Personalized PageRank fingerprints. This framework builds upon other works, including a Bayesian framework for graph matching [6] and an embedding method based on Personalized PageRank [9].

This work's main contributions are:

- A novel node fingerprint definition based on Personalized PageRank, and incorporating anchor nodes, to capture the structural identity of nodes.
- A novel distance metric for calculating similarity between nodes using the introduced node fingerprints.
- A framework for seedless graph matching using the node fingerprints and distance metrics introduced by this work.
- Implementation and evaluation of the framework on synthetic networks using different parameterizations.
- Comparison with another approach for seedless graph matching, SANA [10].

1.3 Organization

The remaining chapters of this dissertation are structured as follows: Chapter 2 introduces relevant concepts and related work within the scope of graph matching and this study. Chapter 3 outlines the contributions of this dissertation, including the proposed framework and implemented algorithm. Chapter 4 describes the proposed methodology and conducts experiments on synthetically generated networks using an edge sampling process with varying edge removal probabilities, and comparing the results with an existing algorithm. Finally, Chapter 5 concludes the dissertation, along with potential future work.

Chapter 2

Background and Related Work

2.1 Graph Matching

Graph matching, also known as network alignment, is a fundamental problem in graph theory and it has applications in various interdisciplinary domains. The goal of this problem is to find a node-to-node correspondence between two graphs. A graph matching solution should be a mapping $\pi : V_1 \to V_2$ which represents the alignment of vertices to maximize the number of correctly mapped edges

There are multiple techniques and approaches for solving the graph matching problem. Typically, the graph structure plays a crucial role, but in some cases, algorithms utilize vertex attributes that are not encoded in the structure. Furthermore, algorithms address the problem in different ways, such as employing the use of seeds, node embedding and heuristics. As examples, [6] employs a framework with Bayesian methods and fingerprints, [10] utilizes simulated annealing heuristics, and [11] proposes a distance based canonical labeling for graph matching.

2.1.1 Isomorphism and automorphism

The graph matching problem is a generalization of the graph isomorphism problem. An isomorphism between two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, is a bijection between their vertex sets, $\pi : V_1 \to V_2$, that preserves adjacency, i.e., an edge $(u, v) \in E_1$ exists if and only if $(\pi(u), \pi(v)) \in E_2$ [12]. Isomorphic graphs are structurally identical, differing only in vertex labels, unlike graphs in a graph matching problem that exhibit similarities we aim to identify.

Determining whether a graph isomorphism exists is an NP problem, and there is no known polynomial-time algorithm for the general case of this problem. However, for certain classes of graphs, such as trees or planar graphs, polynomial-time algorithms do exist.

On the other hand, the concept of graph automorphism refers to an isomorphism from a graph to itself [12]. An automorphism of a graph $G = (V, E)$ is a permutation π of the vertex set V preserving the same set of edges E. Similar to the isomorphism problem, there is no known polynomial-time algorithm for finding automorphisms of a graph for the general case.

The set of all automorphisms of a graph constitutes what is called the automorphism group. Within this set, we can have a collection of vertices that are structurally equivalent in a graph. When addressing the problem of finding a matching π for graph matching, it's important to recognize that multiple valid matchings may exist because of this automorphism group (that can exist in both input graphs), but not all algorithms consider these nuances. Consider that there is an automorphism that maps u to v, where $u, v \in V_2$, in this case it is not possible to distinguish their labels based on the structure, as they are structurally equivalent. Therefore, in this dissertation, when evaluating the framework's performance, if there is a matching $\pi(u') = v$, for $u' \in V_1$ but the correct matching should be $\pi(u') = u$, this will be considered an error in the identified matching. This is because, despite being structurally accurate, the labels do not recognize it as a correct match.

2.1.2 Seeded

Seeded graph matching algorithms use a seed set of pre-mapped node pairs, typically obtained from external information. The seed information is an attribute that is not encoded in the graph structure, except when obtained through a heuristic, in which case the pre-mapped node pairs represent informed estimates. The seed set is typically used as a starting point, where the algorithm then attempts to identify pairs of unmapped nodes that are neighbors of seed nodes, progressively expanding this process to map the entire network.

In the work by Narayanan and Shmatikov [3], an algorithm targeting anonymized social network graphs is implemented. In this algorithm, a graph matching problem is resolved between an auxiliary network and an anonymized network that correspond to real social networks. The work demonstrated that, using a small initial set of seeds, it was able to successfully de-anonymize several thousand users in the anonymous graph.

In the work by Yartseva and Grossglauser [7], as well as the work by Kazemi, Hassani, and Grossglauser [13], the authors utilize seeds and demonstrate the existence of a sharp phase transition. This transition, achieved by increasing the size of the seed set, delineates the boundary between almost certain failure and almost certain success in the percolation graph matching process. These studies formalize the seed set size as a function of network parameters and demonstrate how the size of the seed set can greatly influence the performance of a graph matching algorithm.

2.1.3 Seedless

Unlike seeded algorithms, seedless graph matching algorithms do not require pre-mapped node pairs as input. In this case, the algorithm relies entirely on the network structure, requiring to build the entire mapping.

In the work by Heimann, Shen, Safavi, and Koutra [8], the authors address the graph matching problem using a representation learning approach that does not depend on seed nodes. The proposed algorithm utilizes node embeddings to capture the structural properties of the graphs. These embeddings are then aligned through a process that minimizes the distance between corresponding nodes in the embedding space.

In the work by Pedarsani, Figueiredo, and Grossglauser [6], they propose a Bayesian framework for seedless graph matching with polynomial-time complexity. In their proposed framework, an incremental mapping process is employed. For each graph, nodes are ordered by degree in descending order, with an attempt to first map nodes with higher degrees (which might be easier to differentiate). The execution occurs in phases (or rounds), and in each phase, the nodes that were mapped in the previous phase, called anchors, are then utilized to map additional nodes in the subsequent round.

Furthermore, [6] employs a fingerprint definition, capturing for each node the node degree and hop distances from the node to anchors. Essentially, the method leverages the anchor-mapped pairs to enhance fingerprints and then decide harder matchings. To achieve this, in each round, the number of anchors is doubled, as well as the size of mapped nodes, until the entire network is part of the matching. During each round, the algorithm matches a set of nodes using a maximum bipartite matching. The matching process relies on a measure of similarity (cost function) between two node fingerprints, given by the Bayesian framework that computes the probability that a given pair of nodes is correctly mapped.

The framework proposed in this dissertation draws inspiration from the concepts introduced in [6]. It incorporates similar ideas, including the use of anchor information for fingerprints, execution in rounds, and matching per round using anchors. However, this dissertation distinguishes itself by introducing novel metrics to build the fingerprint and defining new equations for measuring similarity between two fingerprints. Additionally, this dissertation employs a heuristic-based weight function, in contrast to the Bayesian probabilistic model used by Pedarsani, Figueiredo, and Grossglauser.

2.1.4 Node Fingerprint

The node fingerprint comprises a set of attributes, which may include metric values or metric vectors, with the purpose of representing a node. These attributes are designed to capture the node's characteristics based on its topological relationships in the graph.

Depending on the application, the definition of node fingerprint can vary. In [6], the node fingerprint is a set of the node degree and the hop distances between the node and anchor points. In this definition, the fingerprints change with each round due to the new set of anchor nodes.

As a pertinent example for this dissertation, the study in [9] aims to represent nodes using a single vector derived from the intermediate values of the Personalized PageRank algorithm's evolution. This adaptation allows the algorithm to capture the surrounding topology's information, with the vector reflecting the progressive ranking of a node's importance.

In this dissertation, each node fingerprint will consist of a set of vectors related to the execution of the Personalized PageRank algorithm. These vectors are based not only on the vertex that the fingerprint represents but also on the anchor nodes. Consequently, as explored by [6], the fingerprint will change each round due to the new set of anchor nodes.

2.2 PageRank and Personalized PageRank

The PageRank algorithm was first proposed in the work by Brin and Page [14]. It aims to measure the relative importance of web pages based solely on the graph structure, and serves as the foundation for Google's search engine. In this concept, the web is viewed as a graph, where each page represents a node, and the links are directed edges connecting these pages. The importance (rank) of a node is determined by the quantity and quality of the edges pointing to it. In this context, a page's rank depends not only on the number of times it is referenced by other pages but also on the rank of those pages.

The intuition behind PageRank can be illustrated using random walks on a graph. This concept is commonly referred to as "random surfer model", where the surfer navigate through pages by clicking on links at random. In the graph, this represents navigating from node to node according to the edges, and each time a node is visited, its importance increases. To avoid getting trapped in small loops of pages, the surfer occasionally jumps to a random page based on a uniform distribution. The resulting PageRank is determined by the standing probability distribution of this random walk on the graph.

The implementation of the PageRank algorithm involves an iterative process where ranks are updated until convergence. During this process, the graph's random walk transition probability matrix is multiplied by the previous importance vector. The final equation also incorporates the concept of jump (or teleport) based on the uniform distribution. According to this process, it can be noted that for each iteration a certain importance (rank) is assigned to each network node.

The concept of Personalized PageRanks is proposed in the same article [14] and futher explored in [15]. It serves as a method to measure the importance of web pages based on user-specific preferences or specific perspective. To achieve this, the original PageRank algorithm is modified so that the random walker's jumps no longer follow a uniform distribution, but rather a custom distribution. This personalized distribution can either be a probabilistic distribution among a set of specific nodes or always teleport to the same node. Similar to the original algorithm, the iterative process of Personalized PageRank assigns an importance (rank) to each network node at every iteration. Most algorithms are concerned with the converged ranking values. However, this dissertation focuses on the intermediate values obtained during the process to capture information about the graph structure.

As referenced in the work by Gleich [16], the application of PageRank algorithms extends beyond evaluating the importance of web pages within the link structure graph. PageRank can be utilized across any network, and its applications have expanded into multiple domains. These include social network analysis, recommendation systems, and even networks in biology and physics. As previously mentioned, the work conducted in [9] uses PPR values to represent nodes, illustrating the diverse applications that the PageRank algorithm can perform.

Chapter 3

A PPR Framework for Seedless Graph Matching

3.1 General Considerations

The main contribution of this dissertation is a framework for seedless graph matching based on Personalized PageRank (PPR) that uses the structural information of the PPR fingerprint for each vertex to construct a matching between two input graphs. The algorithm works in rounds, where in each round a new matching is determined for the new set of candidate vertices. This matching takes into account the matching of the previous round, even though previously matched nodes can be changed in the current round. The number of candidate vertices increases along with the rounds until it includes all vertices of the input graph with the smallest number of vertices. Algorithm 1 shows the pseudo-code of the framework.

In the following sections, key aspects of the framework are presented, explaining the steps involved and the decisions motivating the design.

3.2 PPR as Fingerprint

The Personalized PageRank (PPR), also called Topic-specific PageRank [14, 15], represents the importance ranking of vertices according to some viewpoint, where viewpoint is a specific set of network vertices. This feature allows generating different rankings for the same network depending on the viewpoint, unlike the original PageRank, where a unique ranking of the vertices is generated for the network.

In this framework, multiple importance rankings will be calculated, each using a single vertex as the viewpoint. This approach allows us to generate unique fingerprints for each vertex based on its viewpoint. Consequently, the PPR will induce in each vertex a different fingerprint based on the importance ranking and the network structure, simply by changing the viewpoint.

Algorithm 1 Framework

```
Input: graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$
Output: matching $\pi$
  $V_1' \leftarrow$ vertices sorted in decreasing order according to structural feature of $G_1$
  $V_2' \leftarrow$ vertices sorted in decreasing order according to structural feature of $G_2$
  round $h \leftarrow 1$
  anchors $S_{h-1} \leftarrow \emptyset$
  final matching size $n \leftarrow min\{|V_1|, |V_2|\}$
  while $true$ do
      $n_h \leftarrow min\{2^h, n\}$
      Define $V_1^h \leftarrow V_1'(1 : n_h)$ and $V_2^h \leftarrow V_2'(1 : n_h)$ as the candidate sets
      Build $G_B \leftarrow G(V_1^h, V_2^h, E^h)$, the weighted complete bipartite graph
      $w_{(u_{1i}, u_{2j})} \in E^h \leftarrow$ weight given by $f_{S_{h-1}}(u_{1i}, u_{2j})$
      $\pi \leftarrow$ (approximate) minimum weight matching for $G_B$
      if $n_h = n$ then
          return $\pi$
      $B_\pi \leftarrow$ vertex pairs (edges) of $\pi$ sorted in ascending order according to $E^h$
      $S_h \leftarrow B_\pi(1 : 2^{h-1})$
      $h \leftarrow h + 1$
```

The PPR importance values can be obtained through an iterative process where, just like in PageRank, after a certain number of iterations converges to the final importance for the vertices. In this dissertation, the initial PageRank importance vector, r_0, will be a column vector with value 1 for the viewpoint vertex and zero for the other vertices. Likewise, the column vector for teleport transitions S will be defined in the same way with value 1 for the viewpoint vertex and zero for the other vertices. Consequently, at $t = 0$ of the iterative process, the importance will be concentrated on the viewpoint vertex. Since the teleport set is defined only as the viewpoint vertex, when considering the iterative process as a random walk, the restart will always be directed to the viewpoint vertex.

Algorithm 2 shows the pseudo-code of the iterative process to calculate the PPR taking into account only a single vertex as the viewpoint. In this iterative process, the importance ranking in a given horizon t, called r_t, will be obtained by multiplying the graph's random walk transition probability matrix M and the previous importance vector r_{t-1}. Applying the teleport concept, using a teleport factor α, we will weight the importance contribution via the random walk matrix and the teleport. The definition of function r_t will be given by:

$$\underset{(vector)}{r_t} = (1-\alpha)Mr_{t-1} + \alpha S \tag{3.1}$$

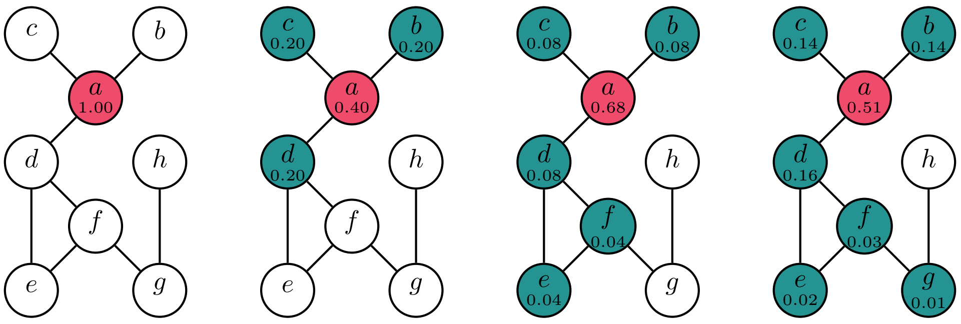
In this framework, we will need the PPR importance values of the iterative process calculated using each of the graph's vertices as a viewpoint. The presented Algorithm 2 can be slightly modified so that in a single execution the values for all vertices are obtained as if the same algorithm had been executed for each different vertex as a viewpoint. Taking Equation 3.1 as a reference, the modified version used in the framework replaces the initial PageRank importance r_0 by an identity matrix and likewise the vector for teleport transition S also by an identity matrix. As a result, r_t will no longer be a vector representing the importance ranking using a given viewpoint vertex on horizon t, but rather a matrix where the values of each column represent the importance ranking for a different vertex as viewpoint on horizon t. The modified definition of function r_t will be given by:

$$\underset{(matrix)}{r_t} = (1-\alpha)Mr_{t-1} + \alpha I \tag{3.2}$$

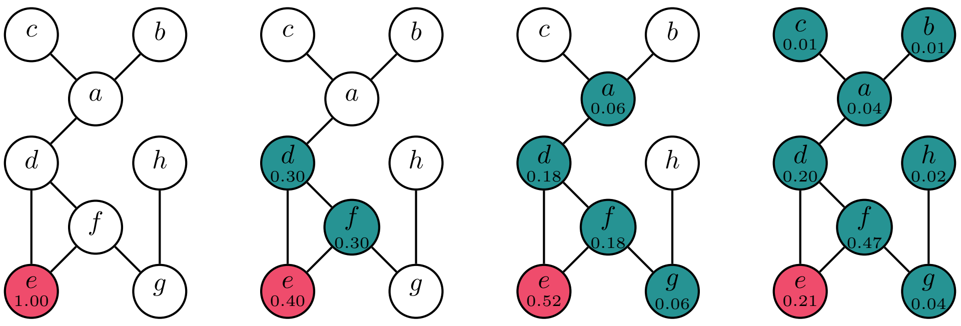
Algorithm 2 Personalized PageRank with single vertex as viewpoint

```
Input: graph $G(V, E)$
Input: viewpoint vertex $u$
Input: teleport factor $\alpha$
Input: horizon of rounds $T$
Output: importance rankings $r_0, r_1, ..., r_T$
  $n \leftarrow |V|$
  $M \leftarrow$ Column stochastic transition probability matrix for $G$, with dimension $n \times n$,
  meaning that if $(j, i) \in E$ and $d_j$ is out-links of $j$, then $M_{ij} = 1/d_j$, else $M_{ij} = 0$.
  $S \leftarrow$ Teleport transition column vector with dimension $n \times 1$, where the only
  non-zero value is at index $u$, with value 1.
  $r_0 \leftarrow$ Initial state column vector with dimension $n \times 1$, where the only non-zero
  value is at index $u$, with value 1.
  $R \leftarrow [r_0]$ importance ranking list
  $t \leftarrow 1$
  while $t \leq T$ do
      $r_t \leftarrow (1-\alpha)Mr_{t-1} + \alpha S$
      add $r_t$ to $R$
      $t \leftarrow t + 1$
  return $R$
```

For some contexts using PPR, only the final importance value of each vertex is relevant (obtained after the algorithm converges), however, the intermediate values obtained during the iterations encode structural information about the network. Thus, in each t of the iterative process, an importance value is given to each vertex of the graph. As a result, the sequence of importance values for a specific vertex allows us to encode more information about the structural role of this vertex than just the final importance value after convergence.



(a) PPR iterative process using a as viewpoint vertex for $t = 0, 1, 2, 3$.



(b) PPR iterative process using e as viewpoint vertex for $t = 0, 1, 2, 3$.

Figure 3.1: Comparison of PPR iterative process using different viewpoint vertices and starting at $t = 0$. In both iterative processes, teleport factor $\alpha = 0.4$ and maximum horizon $T = 3$.

As can be seen in Figure 3.1a starting from a viewpoint vertex of the network, the PPR values expand to the other vertices and if we compare this expansion and the values obtained with Figure 3.1b we can see how the viewpoint changes the PPR values obtained during the iterative process.

We can define a vector $R_u(u)$, or for simplicity R_u, as the sequence of PPR values in the range $0 \leq t \leq T$, where the vertex u is the viewpoint and the vertex of interest at the same time. So in Figure 3.1a we would have $R_a = R_a(a) = [1.00, 0.40, 0.68, 0.51]$ and in Figure 3.1b we would have $R_e = R_e(e) = [1.00, 0.40, 0.52, 0.21]$. Going further, we can think of an additional parameter to access the value of each specific t, so $R_a(a, 0) = 1.00$, $R_a(a, 1) = 0.40$, $R_a(a, 2) = 0.68$ and $R_a(a, 3) = 0.51$. Note that $t = 0$ represents an initial state predefined by the algorithm, and the horizon $T = 3$ (in Figure 3.1) indicates the number of rounds of the PPR iterative process that were executed.

The definition of R_u by itself could be considered a fingerprint for vertex u, as stated in SAB [9]. For this framework we have a new fingerprint definition, that seeks to encode more structural information, and a new metric to calculate the distance between fingerprints.

Analyzing Figure 3.1a and Figure 3.1b and comparing with the previous definition, we can notice that for both $R_a(a)$ and $R_e(e)$ we have the same values for $t = 0$ and $t = 1$. This will always be true when the viewpoint and the vertex of interest are the same and the parameters used to calculate the PPR are also the same. Hence when calculating the distance between these two vectors it makes sense to ignore these values. This will be covered again in Section 3.4.1.

Similarly we can define $R_u(v)$ as the sequence of PPR values where this time vertex u is the viewpoint, but v is the vertex of interest. So in Figure 3.1a we would have, for example, $R_a(f) = [0.00, 0.00, 0.04, 0.03]$, it should be noted that the PPR evolution is still starting from vertex a, but when changing the vertex of interest we have a vector different from R_a. Also, the same viewpoint can generate vectors for all other vertices of interest, so thinking about a vertex u as viewpoint, it is possible to evaluate its structure information on the network with two different perspectives: looking at itself as a vertex of interest through R_u or looking at others vertices as vertices of interest using $R_u(x)|x \in V \setminus \{u\}$.

It is worth noting that when the viewpoint and vertex of interest are different, the vector of $R_u(v)$ will always start with at least one zero. This occurs because the evolution of the PPR from the viewpoint u will take some rounds to reach the vertex of interest v. More specifically, the number of rounds t necessary for the PPR value $R_u(v, t)$ to be non-zero will be exactly the hop distance between u and v in the graph. Thus, depending on the horizon T, the sequence $R_u(v)$ can contain zeros only. Hence when calculating the distance between two vectors, where in each the viewpoint and the vertex of interest are different, it makes sense to consider the precondition of a set of starting zeros that each vector will contain (which can be a sequence of different length). This will be covered in Section 3.4.1.

The algorithm works in rounds and the anchors (seed set) defined in one round will be used to find a new matching in the next round. The set of anchors for $h = 1$, by definition, will be $S_0 = \emptyset$ (empty set), and for subsequent rounds it will have half of the candidate vertices from the previous round that were best evaluated as matching, i.e., the set of anchors that will be used in a round $h > 1$ will be $S_{h-1} = \{a_1, \dots, a_k\}$, where $k = 2^{h-2}$. The framework uses a fingerprint to characterize a vertex and allow two vertices to be compared using their respective fingerprints. Based on this, in each round the fingerprint of the vertices is modified according to the anchors defined in the previous round. The framework defines the fingerprint of a vertex u in a round h, using anchors S_{h-1}, as:

$$F_u = [R_u, R_u(a_1), \ldots, R_u(a_k)] \tag{3.3}$$

Recall that an anchor is a vertex in G_1 that has been matched with a vertex in G_2. From the viewpoint of a vertex u in G_1, the anchors of G_1 provide valuable information for u's fingerprint, indicating the "importance" of each anchor from u's viewpoint. Therefore, using the anchors of G_1 as vertices of interest from u's viewpoint enriches the information about u in G_1. This leads to more informed matching decisions, as the anchors in G_1 have corresponding anchors in G_2, which are used in the fingerprints for vertices in G_2.

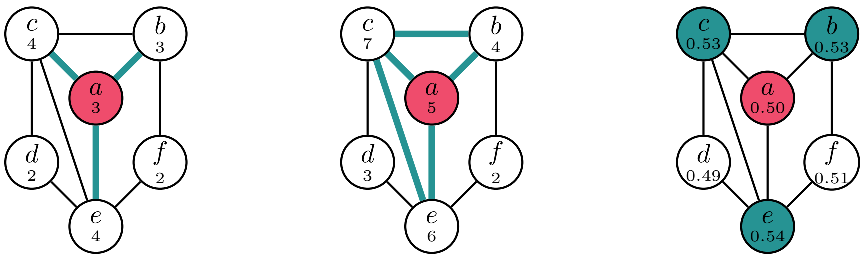
3.3 Ordering Candidate Vertices

In every round of the framework a new matching is determined from a set of candidate vertices chosen from each input graph. In the first rounds the candidate sets are smaller, and in the last round the candidate vertices will calculate the matching between all the vertices of the graph with the lowest cardinality and the same number of vertices from the other graph. Considering graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$, in round h the size of the set of candidate vertices in each graph will be $n_h = \min\{2^h, n\}$, where $n = \min\{|V_1|, |V_2|\}$.

The candidates in each graph are not chosen at random. Initially, for each graph the vertices are sorted in descending order according to some structural feature of the nodes, so in a round h the candidate vertices will be the first n_h of each ordered list. The round's matching will then be calculated between the chosen candidate vertices, therefore, the idea of ordering the vertices is to maximize the chances that the candidate set of one graph contains the corresponding vertices in the candidate set of the other graph. The use of a vertex sorting based on network structural feature and the matching decision using the PPR fingerprint makes the network structure play a central role in this graph matching framework.

A good ordering for candidate vertices should order the corresponding vertices of each graph in the same position (or close to it). In the graph matching problem, the corresponding vertices are in different graphs, which in general have different structures. Therefore, the sorting metric must be sensitive to structure, to encode the vertex correctly in its position, but robust so that small changes (such as the absence of edges or vertices) do not easily affect the ordering, which could prevent candidates from one graph to coincide with the candidates in the other graph.

There are different structural features that could be applied to sort the candidate vertices. In this dissertation we will explore some options to compare how they affect the result of the matchings and which is more reliable according to the experimental evaluation. In Figure 3.2, the following features are presented: degree, edges in EgoNet-1 and the PPR value $R_u(u, 2)$. Considering a vertex u in a network, the degree is simply the number of neighbors of u. For edges in EgoNet-1, we consider an induced subgraph composed of vertex u and all its neighbors and count the edges in this subgraph. Finally, for $R_u(u, 2)$, the PPR iterative process is run for vertex u until $t = 2$ and the value $R_u(u, 2)$ is selected as the node feature for sorting (it is worth noting that $R_u(u, 0)$ and $R_u(u, 1)$ could not be used since each one would have the same value for all vertices of the network).



(a) Degree. (b) Edges in EgoNet-1. (c) $R_u(u, 2)$.

Figure 3.2: Comparison of different structural features used for sorting the candidate set with the value for each vertex. Vertex a is highlighted as are the network properties that influence its value, for 3.2a and 3.2b are edges, and for 3.2c is the importance of neighboring vertices.

By calculating a structural feature for each vertex and using these values to generate an ordering, the list of candidates for each graph will point out differences between the network's vertices, from vertices with stronger characteristics (greater values) to more common vertices (smaller values). The sorting by each of these features can lead to different ordering of the vertices, and will directly affect the results of the framework, since the initial matchings will be executed with different candidate vertices and consequently the anchors that will be used to assist the matchings of subsequent rounds will also be different.

The ordering of candidates can highlight characteristics in complex networks, where some features can present a long-tail behavior. In this case, metrics such as degree can be very unequal between vertices, where the majority may have a small degree and very few have a degree orders of magnitude greater than average. This lack of normality in a feature can benefit the choice of initial candidate sets, but quickly it may not be able to distinguish the vertices with lower values, making the choice of sets more randomized.

It is important to mention that depending on the network, the performance of the same feature may differ, as it is directly associated with its structural characteristics. The idea of analyzing different ordering metrics is not to choose on a case-by-case basis which one performs better, but to have a guideline against the tested examples, and use a metric that proves to be reliable.

Recall that the first round of the framework has no anchors defined, while in the subsequent rounds, rankings using anchors as vertices of interest are added to the fingerprint. Because of this, the initial matching is the least informed and with each new round the matchings become more informed. Starting the matching algorithm by trying to decide the matching between the vertices that can be more easily identified is important because it means choosing good anchors that will provide relevant information for future decisions. Furthermore, when choosing anchors that are not true matches will negatively affect the matchings of subsequent rounds.

3.4 Building the Weighted Graph

The matchings between the vertices of the input graphs, G_1 and G_2, are defined each round from the weighted complete bipartite graph $G_B \leftarrow G(V_1^h, V_2^h, E^h)$ where V_1^h and V_2^h represent the candidate vertices in round h for G_1 and G_2, respectively, and E^h represents the weighted edges between V_1^h and V_2^h. The weight of each edge represents the structural distance between the vertices, where a zero distance represents an identical structure according to the framework's evaluation. Thus, having a reliable distance metric between the fingerprints of two nodes is essential to matching vertices correctly.

A bipartite graph guarantees that a vertex in V_1^h will have its correspondence in V_2^h, i.e., a matching between a vertex in G_1 and a vertex in G_2. Additionally, using a complete bipartite allows a vertex in V_1^h to correspond to any vertex in V_2^h and vice versa. Therefore we know that in a round h, where the candidate set has size n_h, the number of edges in G_B will be $(n_h)^2$, consequently this will also be the number of distances calculated in round h. Figure 3.3 illustrates the complete bipartite graph that is used to determine the matching pairs at round $h = 1$ which has a candidate set of $n_h = 4$ vertices.

3.4.1 Calculating Distances

The weight or distance between two vertices, $w_{(u_{1i}, u_{2j})} \in E^h$, in a round h, is given by $f_{S_{h-1}}(u_{1i}, u_{2j})$. The distance is based on comparing the fingerprint of vertices u_{1i} and u_{2j}, and is composed of two main terms. The first term only takes R_{u1i} and R_{u2j} into account, i.e., without considering anchors. The second term jointly evaluates the anchors through the fingerprint, treating the anchors as vertices of interest. The anchor pairs considered in a round h belong to the set S_{h-1}, defined in round $h-1$, meaning that each pair $(a_{1i}, a_{2j}) \in S_{h-1}$ is composed of $a_{1i} \in V_1^{h-1}$ and $a_{2j} \in V_2^{h-1}$.

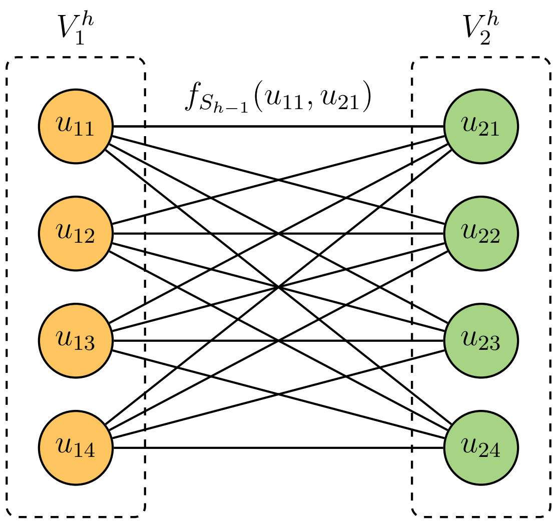


Figure 3.3: Representation of the weighted complete bipartite graph G_B at round $h = 2$ where the candidate set has size $n_h = 4$.

The framework, when calculating the distance between fingerprints, is analyzing different sequences of PPR values in each fingerprint. All sequences of PPR values will have the same length of $T+1$, based on the horizon T parameter, meaning that each sequence has range $0 \leq t \leq T$. As mentioned in Section 3.2 the entire sequence encodes information about the structure, which is why the sequences are analyzed considering the value at each iteration t. However we can think that not every t index is equally reliable.

Thinking about the reliability of iteration t in a PPR sequence, let's imagine the PPR where the viewpoint is a single vertex. During the iterative process, the calculated values expand from the viewpoint vertex towards the others, as can be seen in Figure 3.1. Therefore, when $t = 1$ the vertices with non-zero values will be vertices at hop distance 1 (including the viewpoint), when $t = 2$ all vertices with $d \leq 2$ and so on. Based on this, when $t = 2$, modifications within $d \leq 2$ can affect the calculated values. Likewise, when $t = 3$, changes within $d \leq 3$ will influence the calculated values. Thus, the larger the t, the greater the range that can impact the values. Something similar will also occur when evaluating $R_u(v)$, with the difference that in this case, we will only notice the differences in $R_u(v,t)$ for $t \geq t^*$, where $t^* = \min\{t | R_u(v,t) > 0\} = d_{u,v}$ the hop distance between u and v on the graph.

When comparing two equivalent vertices in different networks we expect the structures to have similarities that will be encoded by the framework. At the same time, since the networks are not identical, there will be structural differences encoded in each fingerprint even for corresponding vertex pairs. As seen previously, with the increase in t the structure evaluated for the PPR also increases, so the greater t, the greater the probability of differences in the structures surrounding corresponding vertex pairs. Based on this, the framework must guarantee a greater or more valuable contribution for smaller values of t, and with each increase in t the contribution added to the comparison must be amortized.

In order to formalize the distance between two vertices, let's define $d(R_{u_{1i}}, R_{u_{2j}})$ as a first distance between two sequences of PPR values for node i in G_1 (denoted u_{1i}) and node j in G_2 (denoted u_{2j}). In this case, for each PPR sequence the viewpoint and the vertex of interest are the same vertex, i.e., no anchors contribute to the distance in this case. The definition of function $d(R_{u_{1i}}, R_{u_{2j}})$ is given by:

$$d(R_{u_{1i}}, R_{u_{2j}}) = \sum_{t=2}^{T} \frac{\left|R_{u_{1i}}(u_{1i}, t) - R_{u_{2j}}(u_{2j}, t)\right|}{\max\left(R_{u_{1i}}(u_{1i}, t), R_{u_{2j}}(u_{2j}, t)\right)} \sigma^{t-2} \tag{3.4}$$

In this metric, at each t analyzed, we calculate the absolute value of the difference between the value of each fingerprint, and normalize it by the maximum between the two values. After that, we apply an amortization according to a factor σ, where σ is within the range $[0, 1]$. The final value will then be a sum of this calculation for $2 \leq t \leq T$, where we start the comparison with $t = 2$ since, as demonstrated in Section 3.2, for $t < 2$ the PPR values will all be the same, and the T horizon is the number of rounds of the PPR iterative process that have been executed. It is worth noting that we use σ^{t-2}, since for $t = 2$ it will be $\sigma^{2-2} = \sigma^0 = 1$, i.e., no amortization and for $t > 2$ this factor will reduce the contribution to the distance (assuming $\sigma < 1$).

The normalization carried out in Equation 3.4 has the purpose of ensuring that for each t the contribution to the distance is proportional to the difference measured, so that it is always within the range $[0, 1]$. At the same time, the amortization used is related to the discussion of reliability of an index t in the PPR sequence, so that the first value considered is not amortized and subsequent values are increasingly amortized.

Similar to Equation 3.4 we define a second distance between two sequence of PPR values, but where this time the viewpoint and vertex of interest are different. In particular, the vertex of interest for each sequence will be anchors. The second distance $d'(R_{u_{1i}}(a_{1k}), R_{u_{2j}}(a_{2k}))$ is defined as:

$$d'(R_{u_{1i}}(a_{1k}), R_{u_{2j}}(a_{2k})) = \sum_{t=t^*}^{T} \frac{\left|R_{u_{1i}}(a_{1i}, t) - R_{u_{2j}}(a_{2j}, t)\right|}{\max\left(R_{u_{1i}}(a_{1k}, t), R_{u_{2j}}(a_{2k}, t)\right)} \sigma^{t-t^*} \tag{3.5}$$

where:

$$t^* = \min\{t | R_{u_{1i}}(a_{1k}, t) > 0 \vee R_{u_{2j}}(a_{2k}, t) > 0\} \tag{3.6}$$

The use of this metric allows the calculation of a distance between u_{1i} and u_{2j}, but taking into account the vertices of interest $a_{1k} \in V_1$ and $a_{2k} \in V_2$ which are corresponding anchors decided in the previous round of the framework. The idea behind the metric is similar to Equation 3.4 but the initial t, as indicated in Equation 3.6, will be the smallest t for which at least one of the values is different from zero (otherwise, both are zero and we have no distance value). Therefore, we also use the same t to start the amortization.

When using Equation 3.5, the anchor vertices, a_{1k} and a_{2k}, are considered a correct matching that will add more information to the distance calculated by the framework. However, when using multiple pairs of anchors there will be pairs with different confidence levels, i.e., anchors that the framework will be more confident of being correct than others. Based on this, we define a confidence metric $w(a_{1k}, a_{2k})$ where the value for each anchor pair is in the range $[0, 1]$, and the sum for all anchor pairs is 1. Greater confidence indicates that the anchor pair is more likely to be a correct match in comparison with the other available anchors. The confidence $w(a_{1k}, a_{2k})$ is defined as:

$$w(a_{1k}, a_{2k}) = 1 - \frac{d(R_{a_{1k}}, R_{a_{2k}})}{\sum_{(a_{1x}, a_{2x}) \in S_{h-1}} d(R_{a_{1x}}, R_{a_{2x}})} \tag{3.7}$$

Once we have a distance definition for two vertices using a single anchor given by Equation 3.5, the next step is to define a distance between two vertices considering a set of anchor pairs. For this we will combine Equations 3.5 and 3.7 to weight the contribution of each anchor, therefore anchor pairs with higher confidence will contribute more compared to pairs with lower confidence. The distance considering a set of anchors is defined by the function $d^a(u_{1i}, u_{2j}, S_{h-1})$ as follows:

$$d^a(u_{1i}, u_{2j}, S_{h-1}) = \sum_{(a_{1k}, a_{2k}) \in S_{h-1}} d'(R_{u_{1i}}(a_{1k}), R_{u_{2j}}(a_{2k}))w(a_{1k}, a_{2k}) \tag{3.8}$$

The framework's final distance given to a pair of vertices and considering the set of anchors from the previous round will be based on Equations 3.4 and 3.8. To combine these metrics, each component will be normalized by its maximum value among the distance of other pairs of vertices and finally the components will be added taking into account γ as a weighting parameter, where γ is within the range $[0, 1]$. The final distance is defined by $f_{S_{h-1}}(u_{1i}, u_{2j})$ as given by the following equations:

$$\bar{d}(R_{u_{1i}}, R_{u_{2j}}) = \frac{d(R_{u_{1i}}, R_{u_{2j}})}{\max\{d(R_{u_{1z}}, R_{u_{2q}}), \forall(u_{1z} \in V_1^h, u_{2q} \in V_2^h)\}} \tag{3.9}$$

$$\bar{d^a}(u_{1i}, u_{2j}, S_{h-1}) = \frac{d^a(u_{1i}, u_{2j}, S_{h-1})}{\max\{d^a(u_{1z}, u_{2q}, S_{h-1}), \forall(u_{1z} \in V_1^h, u_{2q} \in V_2^h)\}} \tag{3.10}$$

$$f_{S_{h-1}}(u_{1i}, u_{2j}) = \bar{d}(R_{u_{1i}}, R_{u_{2j}})\gamma + \bar{d^a}(u_{1i}, u_{2j}, S_{h-1})(1-\gamma) \tag{3.11}$$

The use of normalization by the maximum value among other pairs of vertices in Equations 3.9 and 3.10 ensures that each component of the final distance between two vertices is within the range $[0, 1]$. This allows the weight parameter γ to effectively control the contribution of each component. It can be noted that for $\gamma = 1$, the final distance relies solely on the distances without considering the anchors. This implies that the fingerprints defined for each vertex would function as if no anchors were used. Therefore, executing the process in rounds with this parameterization, unlike rounds with anchors, does not enhance the information of each vertex for more informed matching decisions.

The use of σ in Equations 3.4 and 3.5 as well as γ in Equation 3.11 allows fine-tuning the distance metric. The σ is associated with amortization when evaluating higher t indexes in a PPR sequence and γ controls the relative importance between PPR metrics considering or not considering the set of anchors. Although these are the parameters directly associated with the distance calculation, the parameters used to calculate the PPR, as described in Section 3.2, play a central role in the final distance given by $f_{S_{h-1}}(u_{1i}, u_{2j})$.

The final distance between two vertices is determined by the unique fingerprint of each vertex. The anchors, defined in each round, are essential in shaping the fingerprint and, as a result, affect the distance calculations. Therefore, the distance between any two vertices is not static but varies with each round of the process.

3.5 Minimum Weight Matching in Bipartite Graph

After building the weighted graph, $G_B \leftarrow G(V_1^h, V_2^h, E^h)$, we have by definition the distances between each pair of all possible matchings. Thus, finding the minimum weight matching in the bipartite graph means finding the combination where pairs of vertices are as close as possible.

The problem of finding the minimum weight matching in bipartite graph is itself another problem in computer science. This problem can be stated as finding a perfect matching M minimizing $w(M) = \sum_{e \in M} w(e)$, where $w(M)$ is the total weight of M and $w(e)$ is the weight of edge e. A perfect matching is a subset of edges where no two edges share a vertex and every vertex of the graph belongs to exactly one edge.

For the framework, an algorithm that finds the optimal solution will be equivalent to finding the minimum distance sum where each vertex $u \in V_1^h$ has one and only one corresponding vertex in V_2^h, and the same applies for each vertex $v \in V_2^h$ to have exactly one corresponding vertex in V_1^h. There are multiple algorithms that can be used to solve this problem, one of them is the Hungarian Algorithm that finds the optimal solution in polynomial time, with a time complexity of $\mathcal{O}(n^3)$ in the case of complete bipartite graphs with n vertices.

At each round, with the weighted bipartite graph built, the framework calls a separate module to solve the minimum weight matching. With this setup, the framework can use any algorithm to find a perfect matching in the bipartite graph. For this dissertation, a greedy algorithm was implemented to generate an efficient approximation of the optimal solution. In this greedy algorithm the edges are sorted in ascending order by weight and then each edge in the order is added to the matching if it does not violate the requirements to be a matching. The time complexity of this approach is $\mathcal{O}(n^2 \log n)$ which is the time required to sort all edges of the complete bipartite graph.

Algorithm 3 Greedy Minimum Weight Matching

```
Input: weighted edges $E$
Output: perfect matching $M$
  $E' \leftarrow$ edges sorted in ascending order according to $w(e)$ for each $e \in E$
  $M \leftarrow \varnothing$ empty list of matching edges
  $A \leftarrow \{\}$ empty hash set of vertices used in partition A
  $B \leftarrow \{\}$ empty hash set of vertices used in partition B
  $i \leftarrow 1$
  while $i \leq |E'|$ do
      $(a, b) \leftarrow E'[i]$
      if $a \notin A$ and $b \notin B$ then
          adds $a$ to $A$
          adds $b$ to $B$
          adds $(a, b)$ to $M$
      $i \leftarrow i + 1$
  return $M$
```

During the execution of the framework, only in the last round we will be interested in all the edges found by the minimum weight matching. Therefore, in Algorithm 3, since it builds the matching starting with those with greater similarity (edge with the lowest weight), a small optimization is to use a new parameter to indicate the size of matching to be returned, and thus stopping early the matching procedure. The algorithm could then monitor whether this condition has already been met to return the partial matching before finding a perfect matching.

3.6 Implementation

All framework algorithms were designed and implemented using Python 3. The implementation of the framework can be divided into five modules: coordinator, which coordinates execution; sorting metric, which implements the sorting criteria for candidate vertices; PPR, responsible for calculating the PPR metric for all vertices of both input graphs; distance metrics, which computes the vertex fingerprints and their distances; and minimum weight matching algorithm, responsible for finding the matching in the bipartite graph.

During initialization, the framework shuffles all the vertices of each input graph and then maps them to a new vextex identifier (e.g., vertex number). Due to this, the framework's algorithms only know the generated ids of the vertices. The idea prevents the framework from being biased by using the original ids of the vertices, especially when analyzing synthetic data, where the input graphs may use the same identifier for vertices that are the correct match. Due to shuffling, even if a tie occurs when analyzing a metric, for example during the ordering of candidate vertices, the framework will not be biased in ordering the corresponding vertices in the same order position.

The framework receives a set of parameters required by each its modules. To define the order of candidate vertices, it receives the sort method that must be adopted. In the case of PPR, the parameters provided are the teleport factor α and the maximum horizon T of iterations to be calculated. For distance metrics, the required parameters are the amortization factor σ and the contribution weight of the distance component with no anchors γ. In the case of minimum weight matching, there is no parameter, but the implemented module uses a greedy algorithm.

Regarding implementation details, PPR results are obtained and stored in a sparse matrix, since depending on the size and diameter of the graph and the maximum horizon T, space could easily be a limitation. The PPR matrix for each iteration is saved so that it can later be queried by distance metrics. Note that for $t = 0$ the matrix is not stored, since the PPR values do not depend on the graph structure or parameters, and only on the viewpoint and vertex of interest.

The framework algorithm is a CPU bound process, where most of the execution time occurs calculating the weighted bipartite graph. Based on this, to improve the framework's performance, during distance calculations the framework makes use of process-based parallelism, available with Python's multiprocessing module, and using `concurrent.futures.ProcessPoolExecutor` which offers a higher level interface for submitting tasks.

The framework's code, along with the datasets used for evaluation, is publicly available at [17].

3.7 Time-Complexity Analysis

In this section, we evaluate the time complexity of the framework step by step, and finally, its overall time complexity. To simplify the notation, let's assume that both graphs have $|V_1| = |V_2| = n$ nodes.

The proposed framework includes steps that are executed only once as part of the preparation. Additionally, there are processes that are executed in each round of the framework until the final result is achieved.

Preparatory steps for the framework:

- Sorting vertices: The vertices of each input graph are sorted according to a structural feature. The time-complexity of performing the sorting twice (a constant factor) is $\mathcal{O}(n \log n)$.
- Execution of PPR: The computation of PPR values, as defined using Equation 3.2, involves iteratively multiplying the column stochastic transition probability matrix M (of dimension $n \times n$) by the importance matrix r (of dimension $n \times n$) for T iterations. Given that T is a constant, the time complexity of this process is $\mathcal{O}(n^3)$.

Steps performed in each round of the framework:

- Minimum Weight Matching: In the bipartite graph, it is necessary to determine the matchings for each round. Using the Greedy Algorithm 3, the time complexity will be $\mathcal{O}(k^2 \log k)$, where k is the number of vertices corresponding to one of the candidate sets. For the final round, $k = n$, resulting in a time complexity of $\mathcal{O}(n^2 \log n)$.

Since the framework operates in $\log_2 n$ rounds, we need to consider the complexity of the minimum weight matching being executed for each round. Consequently, the final time complexity for the framework will be $\mathcal{O}(n^2 \log^2(n))$ considering the execution phase only and $\mathcal{O}(n^3)$ for the preparation phase.

Chapter 4

Methodology and Evaluation

4.1 Methodology

The methodology described in this chapter evaluates the framework by employing two types of random graph models: Erdős-Rényi [18] and Barabási-Albert [19]. The goal is to assess and characterize the algorithm's effectiveness in the context of graph matching. To achieve this, the evaluation process involves executing the framework algorithm having as input two distinct but structurally similar graphs, denoted as G_1 and G_2.

To generate the input graphs, we begin by creating an initial graph G_0 using the desired random model. This initial graph serves as the foundation for generating two distinct graphs, G_1 and G_2, through an edge sampling process. The edge sampling process runs independently to generate G_1 and G_2 evaluating each edge of the initial graph and removing it with probability ρ (removing edge probability). As a result, the process will generate two distinct graphs, with some shared and some unique edges. The probability of an edge u of G_0 being present in both G_1 and G_2 is $(1-\rho)^2$.

In this dissertation, 80 pairs of graphs, G_1 and G_2, were generated. To test the performance of the framework against different types of networks, different attributes were used to generate the instances and for each selected set of attributes, 10 instances were generated according to Table 4.1. The attributes include: generative model adopted, Erdős-Rényi or Barabási-Albert; graph size, $n = 512$ or $n = 1024$; and removing edge probability, $\rho = 0.1$ or $\rho = 0.2$. It is worth mentioning that regardless of the generative model adopted, it was parameterized to generate networks with average degree $\bar{d} = 2\log_2(n)$.

The generated instances serve as input for the framework, and will be evaluated with parameter variations. The idea of comparing different framework parameters is to see how they impact the result and evaluate whether any parameterization stands out compared to others.

Model	n	ρ	Instances
Erdős-Rényi	512	0.1	10
Erdős-Rényi	512	0.2	10
Erdős-Rényi	1024	0.1	10
Erdős-Rényi	1024	0.2	10
Barabási-Albert	512	0.1	10
Barabási-Albert	512	0.2	10
Barabási-Albert	1024	0.1	10
Barabási-Albert	1024	0.2	10

Table 4.1: List of instances generated to evaluate the framework's performance. Each instance is composed of G_1 and G_2 generated through the edge sampling process under a graph G_0.

In the context of the framework, we consider parameters related to its implementation and those related to the execution of the PPR. For this evaluation, we kept the PPR-related parameters constant while introducing variations to some of those specific to the framework itself. The following definition describes these parameters:

- PPR teleport factor: $\alpha = 0.5$ (see Equation 3.2)
- PPR horizon of rounds: $T = 5$ (see Algorithm 2)
- Framework amortization factor: $\sigma = 0.9$ (see Equations 3.4 and 3.5)
- Framework candidate sorting feature: parameterized by execution (refer to Section 3.3)
- Framework final weighting parameter γ: parameterized by execution (see Equation 3.11)

As the framework of this dissertation works in rounds, in addition to evaluating the final result of the matching found, this also allows us to evaluate the evolution of the results obtained round by round. In each round, the maximum number of hits that the framework could make would be the size of the set of candidates, but if not every vertex of a candidate set has a corresponding vertex in the other candidate set, this number will necessarily be smaller. Evaluate the evolution of the framework's performance allows us to obtain insights into the functioning of the framework and the evolution of the choice of anchors round by round.

As a comparison model, the instances were also submitted to SANA [10] (Simulated Annealing Network Aligner), publicly available at [20]. The graphs were provided as input in the form of edge lists, where the order of the edges has been intentionally shuffled. The only parameterization provided to the algorithm was `-s3 1`, which determines the weight of the objective function (s3 refers to Symmetric

Substructure Score). As we only have the algorithm's matching output result, we compared only the final result achieved.

In each input instance, let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be graphs with corresponding vertices that share the same label. The score evaluation in the graph matching problem is then obtained by comparing the number of vertex pairs identified by the framework as matching and having the same label, relative to the total number of pairs. Therefore, in a round h, where the framework returns a mapping $\pi : V_1^h \to V_2^h$ that maps the vertices of G_1 to G_2, with $|V_1^h| = |V_2^h| = n_h$, the score for π in round h, denoted as s_π^h, can be defined by the indicator function 4.1 and Equation 4.2 as follows:

$$\delta(a,b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b \end{cases} \tag{4.1}$$

$$s_\pi^h = \frac{\sum_{u \in V_1^h} \delta(\pi(u), u)}{n_h} \tag{4.2}$$

Given this, the score does not take into consideration nodes in V_1 or V_2 that have a non-trivial automorphism. Recall that if there exists an automorphism that maps two or more nodes to each other, these nodes are indistinguishable by the structure of the graph. In this way, nodes with the same structure but different labels can be counted as mismatches. For example, assuming there exists an automorphism that maps u to v, where $u, v \in V_2$, and $u' \in V_1$ corresponds to u in V_2 (therefore sharing the same label), but the framework returns $\pi(u') = v$, this will be counted as an error in the metric being used, despite being structurally accurate.

4.2 Results on Erdős-Rényi graphs

The Erdős-Rényi [18] model, also known as the $G(n,p)$ model, generates random graphs with n nodes by connecting each possible pair of nodes with a probability p independently. This is a classic random model where the structure is random and depends on the execution. Using n and p as parameters, any graph with n vertices can be generated by the model, however the probability of generating each graph will be different depending on p.

Compared to real networks, such as social networks, $G(n,p)$ exhibits lower clustering and the degree distribution is not heavy-tailed. Unlike what is expected in real networks, the degree distribution is a binomial concentrated around the average degree. As the model is parameterized with n and p, it will obey the number n of vertices and will present the average degree according to p.

In this model, the degree of each vertex results from the sum of $n-1$ independent random variables representing the possible edges. Therefore, to obtain the average degree $\bar{d} = 2\log_2(n)$, $p(n) = 2\log_2(n)/(n-1)$ was used as parameter.

The results obtained for $n = 512$ are available in Appendix A, and are qualitatively similar to the results presented here for $n = 1024$.

4.2.1 Evolution

Using the $G(n, p)$ model, we first evaluate the candidate sorting features of degree, edges in EgoNet-1 and the PPR value $R_u(u, 2)$. For all features we use the same weighting parameter $\gamma = 0.25$, i.e., in the final distance 0.25 will be the contribution of the metric not considering the anchors, and 0.75 will be the contribution of the metric considering the anchors.

In Figure 4.1 and 4.2, the graphs with $n = 1024$ using $\rho = 0.1$ and $\rho = 0.2$ are evaluated respectively. The figures show the performance of the framework round by round, on average for the 10 instances with this set of attributes, and according to the candidate sorting feature.

It is possible to see that EgoNet-1 as the ordering for the candidate set outperforms the others features. Using this sorting approach, the accuracy reaches 100% for $\rho = 0.1$ and exceeds 50% for $\rho = 0.2$. Additionally, for both values of ρ, it is possible to notice that the performance for EgoNet-1 exhibits a significant upward trend after a certain number of iterations. Specifically, for $\rho = 0.2$, this increase occurs after a substantial decrease in the number of hits.

Analyzing the use of other sorting features, we notice that degree has an average

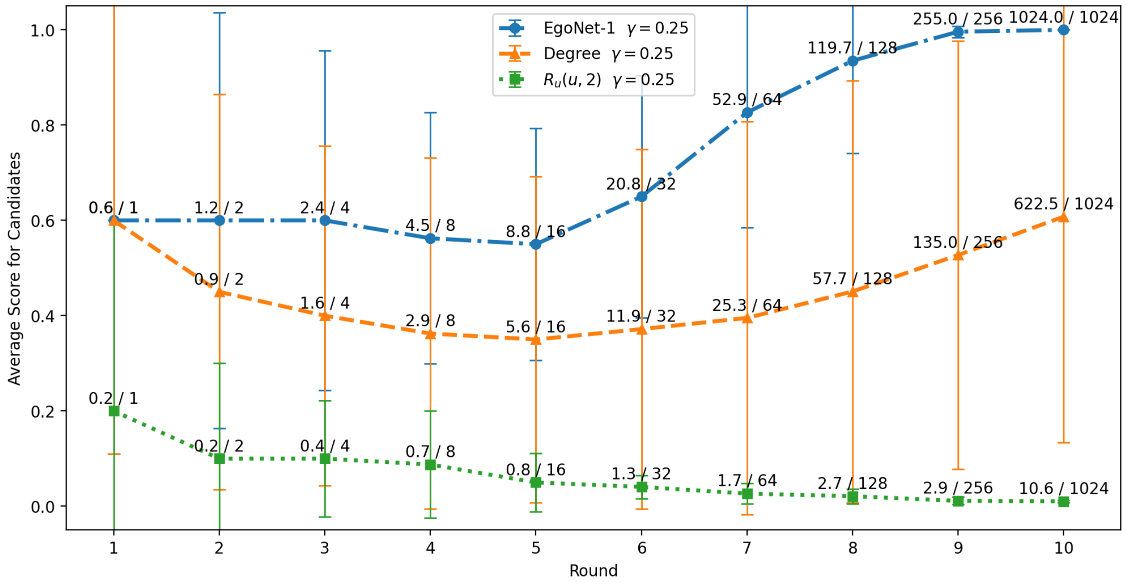


Figure 4.1: Average framework round by round score with standard deviation for the 10 graph instances generated by the $G(n, p)$ model with $n = 1024$ and $\rho = 0.1$. Framework execution comparing structural features for sorting the candidate set.

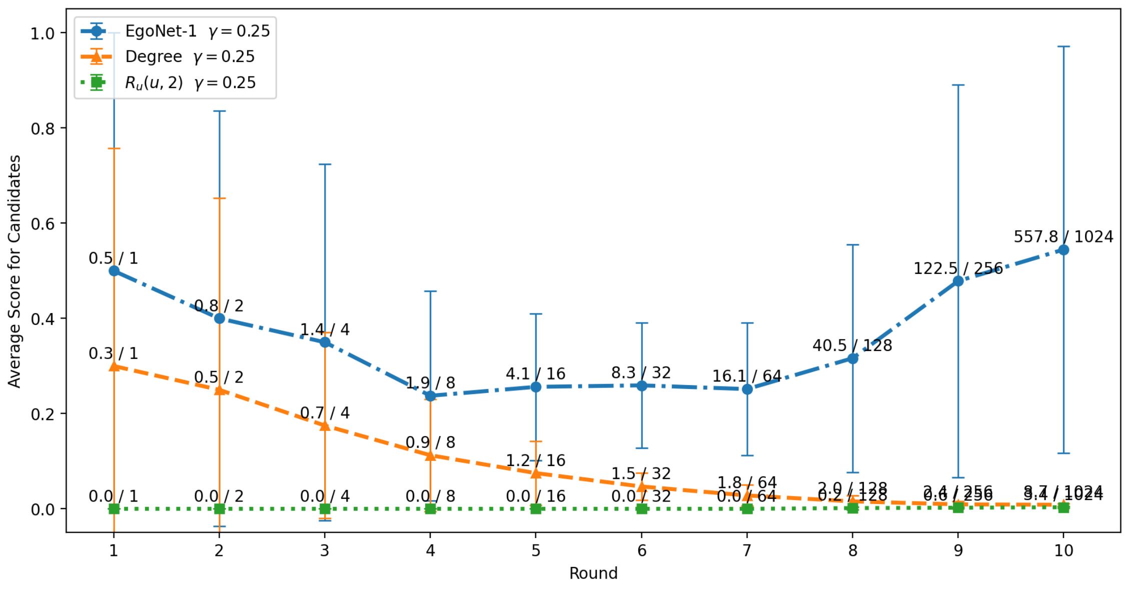


Figure 4.2: Average framework round by round score with standard deviation for the 10 graph instances generated by the $G(n, p)$ model with $n = 1024$ and $\rho = 0.2$. Framework execution comparing structural features for sorting the candidate set.

final performance intermediate between EgoNet-1 and $R_u(u, 2)$ for $\rho = 0.1$, starting out the same as EgoNet-1 and reaching a final result of over 50% accuracy. However, for $\rho = 0.2$, although it initially demonstrates superior performance compared to $R_u(u, 2)$, it ultimately converges to a similar performance level, with an accuracy of less than 1%.

Given the higher performance of EgoNet-1 as the sorting feature, we compare the performance of different weighting parameter γ keeping the sorting feature fixed as EgoNet-1 and varying only γ. In Figure 4.3 and 4.4, the graphs with $n = 1024$ using $\rho = 0.1$ and $\rho = 0.2$ are evaluated respectively. The figures show the performance of the framework round by round, on average for the 10 instances with this set of attributes, and according to the weighting parameter.

Analyzing the performance in the instances with $\rho = 0.1$, as shown in Figure 4.3, it is noted that the influence of γ is not substantial, with the occurrences varying in which one has a better performance but maintaining the same trend, and in the end all obtaining the maximum score.

On the other hand, in the instances with $\rho = 0.2$, Figure 4.4, we observe significant differentiation as we approach the final iterations. When $\gamma = 0.15$ the performance drops below 40% and for the values of $\gamma = 0.10$, $\gamma = 0.20$, $\gamma = 0.25$, the performances converge, ranging between 54% and 59%.

In the assessments using EgoNet-1 as sorting feature, it can be observed a tendency for a drop in accuracy in the initial iterations, more evident for $\rho = 0.2$, followed by an upward trend around the middle round of the total number of iterations. This upward trend may be associated with the use of more anchors that will contribute to the distance, where these anchors among themselves will also have their confidence evaluated to contribute more or less to the distance metric.

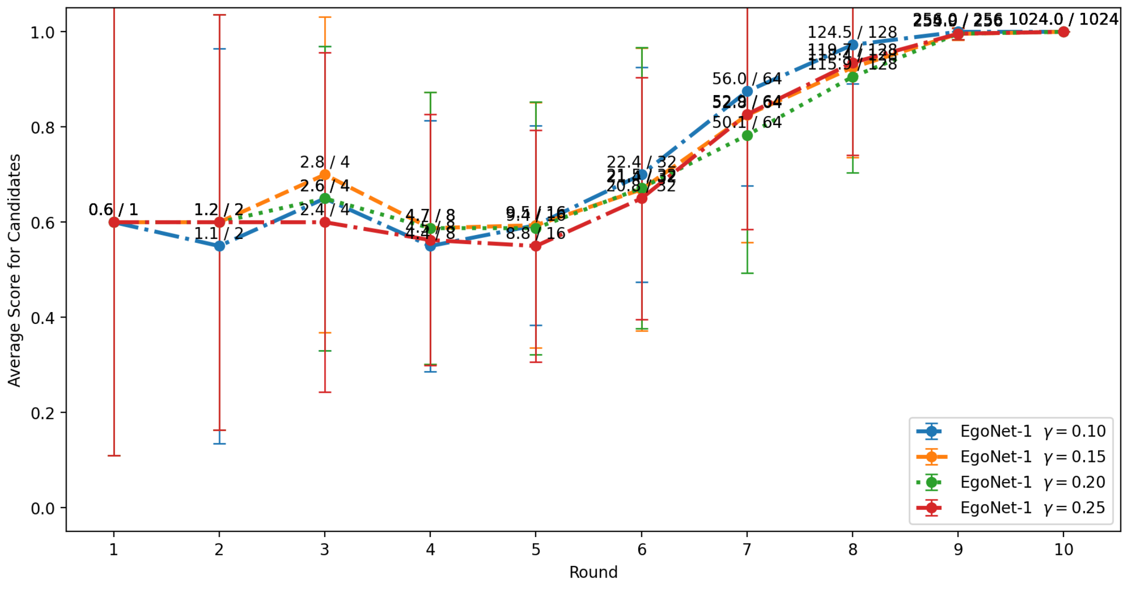


Figure 4.3: Average framework round by round score with standard deviation for the 10 graph instances generated by the $G(n, p)$ model with $n = 1024$ and $\rho = 0.1$. Framework execution using edges in EgoNet-1 and comparing values for the weighting parameter γ.

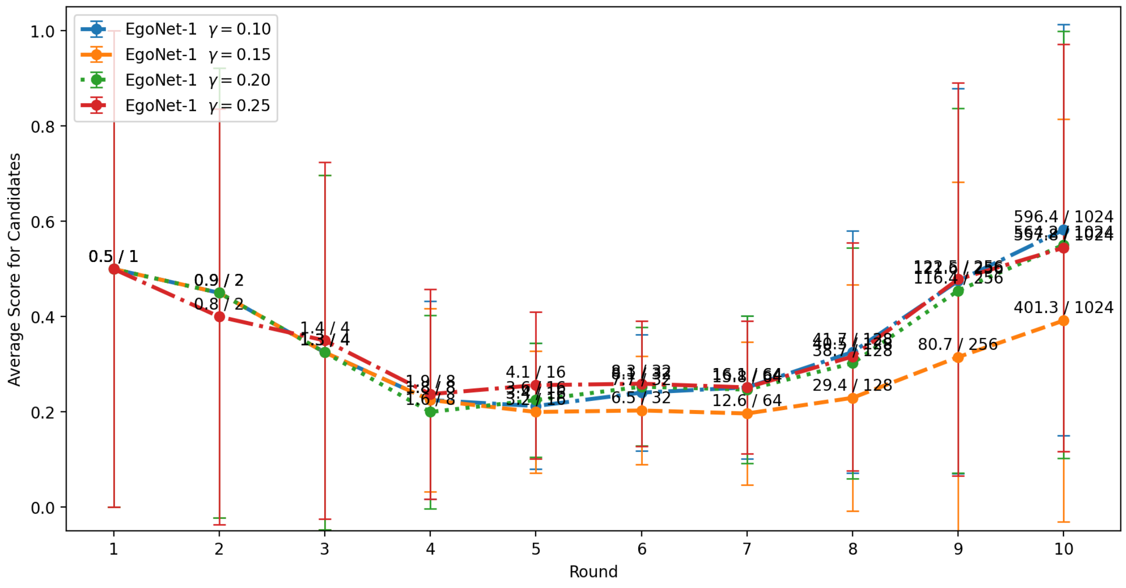


Figure 4.4: Average framework round by round score with standard deviation for the 10 graph instances generated by the $G(n, p)$ model with $n = 1024$ and $\rho = 0.2$. Framework execution using edges in EgoNet-1 and comparing values for the weighting parameter γ.

4.2.2 Matching Result

In Figure 4.5 and 4.6 we have the comparison of the final matching result for the framework and for the SANA algorithm for $\rho = 0.1$ and $\rho = 0.2$, respectively. In both comparisons we have the results per instance and on average. In the case of the framework we have the results for edges in EgoNet-1 as the sorting feature, with variations of the weighting parameter γ, and for SANA the simple execution without trying to change parameterizations.

Comparing the final matching results obtained by the framework, both per instance and on average, the outcome at $\rho = 0.1$ demonstrates equal performance across all instances with 100% accuracy. This performance indicates that even with varying γ values, the framework achieves excellent results for a non-aggressive edge sampling process. On the other hand, for the SANA executions in all instances, the results were below 1% accuracy (ranging from 0 to 2 correct matches out of the 1024 possible pairings). The SANA results may be attributed to the $G(n, p)$ model, he algorithm may not be well-suited for it, given the differences from the results that will be presented later for the BA model.

For graphs with $\rho = 0.2$, the results vary depending on the instance and the γ adopted for the framework, while consistently remaining below 1% accuracy for SANA (ranging between 0 and 4 correct matches out of the 1024 possible pairings). In some instances, regardless of γ, the framework produces similar results, such as for instances 1, 2, 8, and 9. However, for other instances, a specific γ value stands out. For example, in instance 5, $\gamma = 0.25$ significantly outperforms other values, and in instance 10, $\gamma = 0.10$ excels. Notably, edge sampling with $\rho = 0.2$ shows that framework results are instance-dependent, with some results being very low and others achieving 100% accuracy. On average, γ values of 0.10, 0.20, and 0.25 yield the best results, ranging between 54% and 58% accuracy.

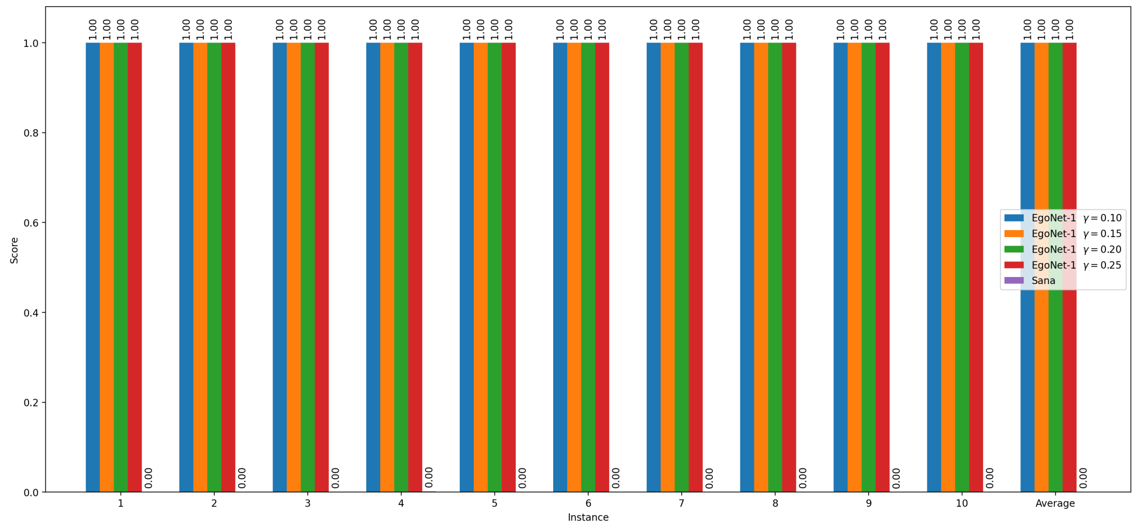


Figure 4.5: Performance comparison for each instance and on average for different parameterizations of the framework and the SANA algorithm. Executions under the 10 instances generated by the $G(n,p)$ model using $n = 1024$ and $\rho = 0.1$.

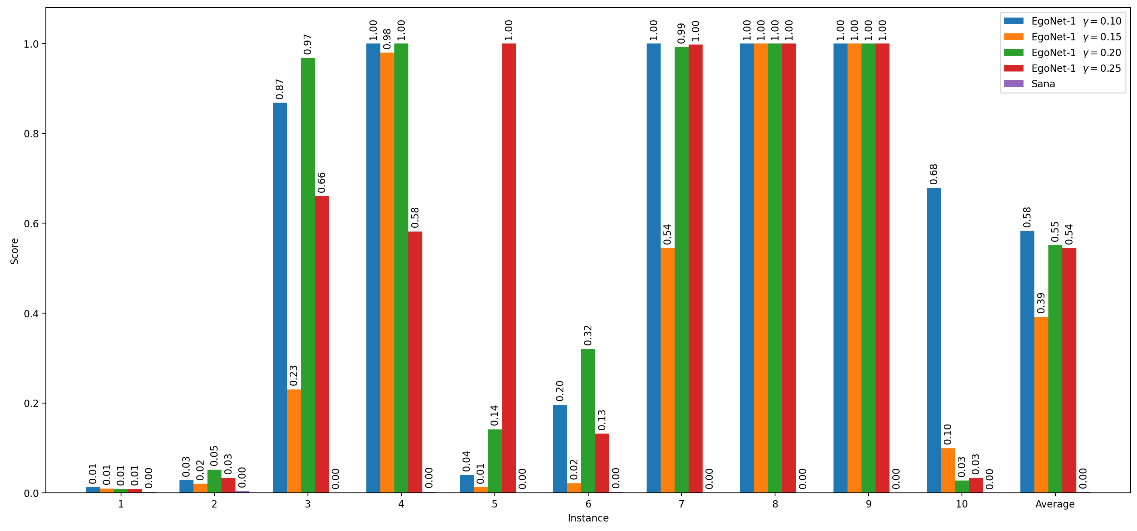


Figure 4.6: Performance comparison for each instance and on average for different parameterizations of the framework and the SANA algorithm. Executions under the 10 instances generated by the $G(n,p)$ model using $n = 1024$ and $\rho = 0.2$.

4.3 Results on Barabási-Albert graphs

The Barabási–Albert (BA) [19] model creates scale-free networks using a preferential attachment mechanism. In this model, new nodes are added over time and are more likely to connect to existing nodes with higher degrees, simulating real-world networks with power-law degree distributions, where some nodes (hubs) have significantly more connections.

This model starts with a small clique graph and at each step a new vertex is added with degree m. Thus, the degree distribution in the BA model is a heavy-tailed distribution, unlike the $G(n, p)$, but the building process also does not favor the existence of triangles and consequently the generated network has low clustering.

The input parameters of this model are n and m, where n is the number of vertices in the final network, and the average degree will be determined according to m, so that $\bar{d} = 2m$. Therefore to obtain the average degree $\bar{d} = 2 \log_2(n)$, the same as was used for the $G(n, p)$ instances, $m = \log_2(n)$ was adopted.

The results obtained for $n = 512$ are available in Appendix B, and are qualitatively similar to the results presented here for $n = 1024$.

4.3.1 Evolution

As assessed for the other model, using the BA model we first evaluate the candidate sorting features of degree, edges in EgoNet-1 and the PPR value $R_u(u, 2)$. For all features we use the same weighting parameter $\gamma = 0.25$, i.e., in the final distance 0.25 will be the contribution of the metric not considering the anchors, and 0.75 will be the contribution of the metric considering the anchors.

In Figure 4.7 and 4.8, the graphs with $n = 1024$ using $\rho = 0.1$ and $\rho = 0.2$ are evaluated respectively. Similar to the evaluation for the $G(n, p)$ model, it is observed that EgoNet-1 outperforms the other features as the ordering for the candidate set. However, for the BA model, in instances with $\rho = 0.1$, the degree feature also achieves 100% accuracy. In instances with $\rho = 0.2$, EgoNet-1 achieves superior results compared to the same ρ value in the $G(n, p)$ model, obtaining over 80% accuracy for the BA model, while the degree feature exhibits intermediate performance between EgoNet-1 and $R_u(u, 2)$ with over 60% accuracy.

Following the same procedures adopted for the $G(n, p)$ model, due to the superior performance of the EgoNet-1 feature, we compare the performance of different weighting parameter γ keeping the sorting feature fixed as EgoNet-1 and varying only γ. In Figure 4.9 and 4.10, the graphs with $n = 1024$ using $\rho = 0.1$ and $\rho = 0.2$ are evaluated respectively.

Similar to the $G(n, p)$ model, adjusting the γ parameter in the BA model with $\rho = 0.1$ slightly influences the score evolution during framework iterations, and these iterations ultimately converge to the maximum score. For instances with $\rho = 0.2$, the behavior of accuracy initially decreasing followed by an upward trend becomes more pronounced. In this context, the instances exhibit similar performances, but as we approach the final iterations, they differentiate. Specifically, γ values of 0.20 and 0.15 yield the highest accuracies (97% and 90%, respectively), while γ values of 0.10 and 0.25 result in accuracies 85% and 83%, respectively.

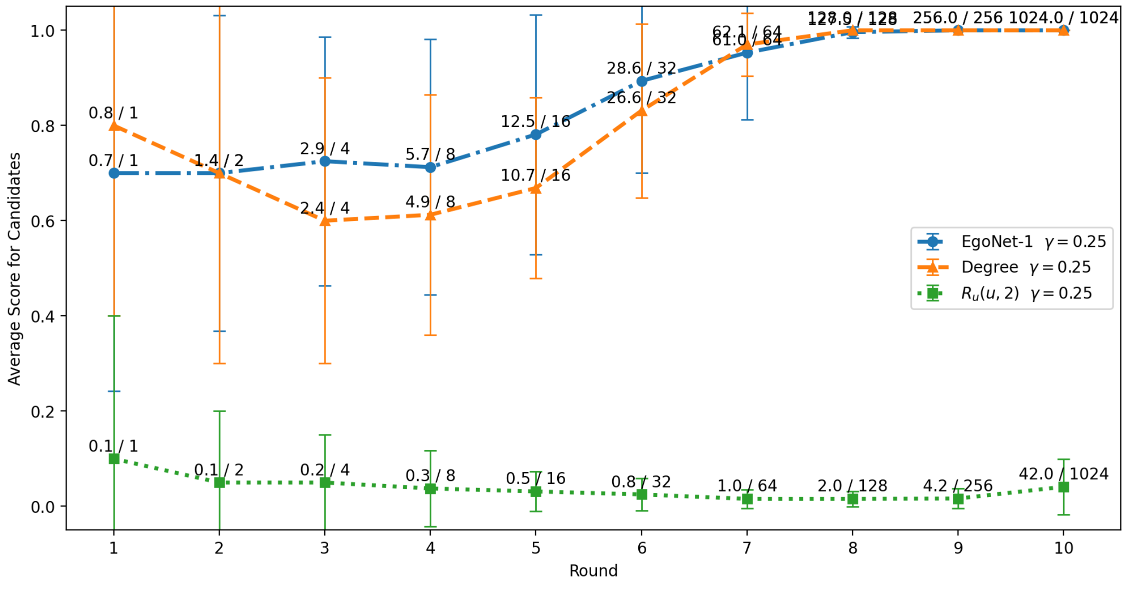


Figure 4.7: Average framework round by round score with standard deviation for the 10 graph instances generated by the BA model with $n = 1024$ and $\rho = 0.1$. Framework execution comparing structural features for sorting the candidate set.

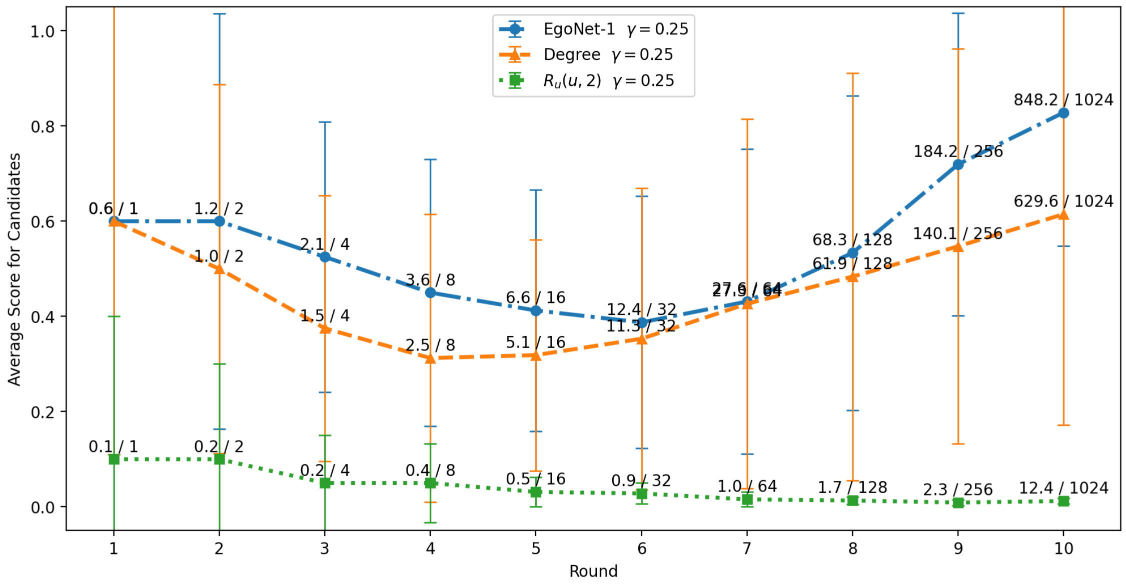


Figure 4.8: Average framework round by round score with standard deviation for the 10 graph instances generated by the BA model with $n = 1024$ and $\rho = 0.2$. Framework execution comparing structural features for sorting the candidate set.

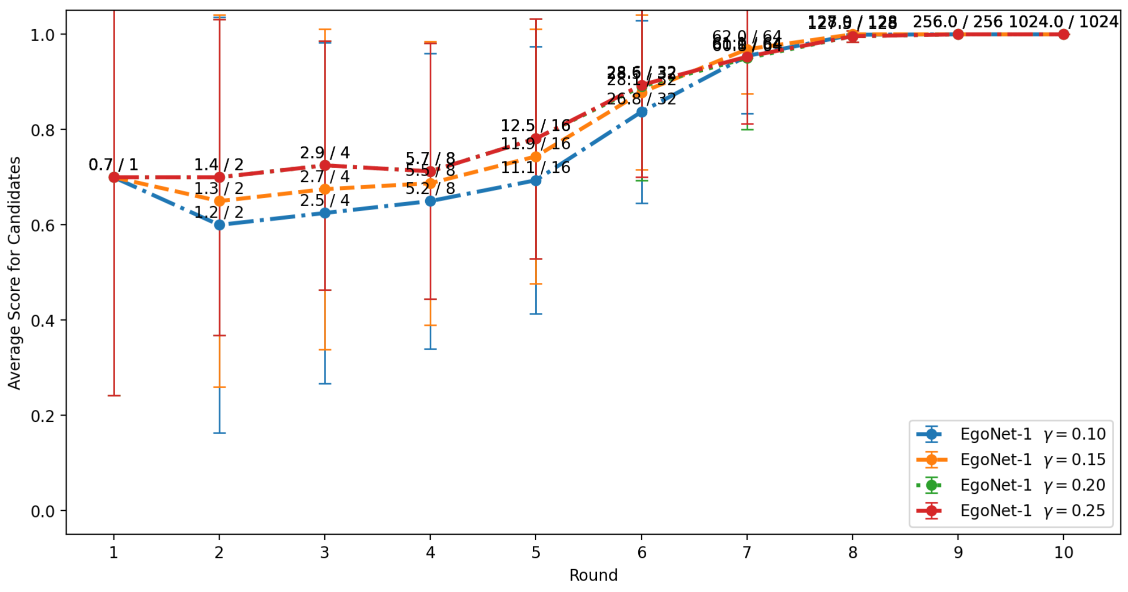


Figure 4.9: Average framework round by round score with standard deviation for the 10 graph instances generated by the BA model with $n = 1024$ and $\rho = 0.1$. Framework execution using edges in EgoNet-1 and comparing values for the weighting parameter γ.

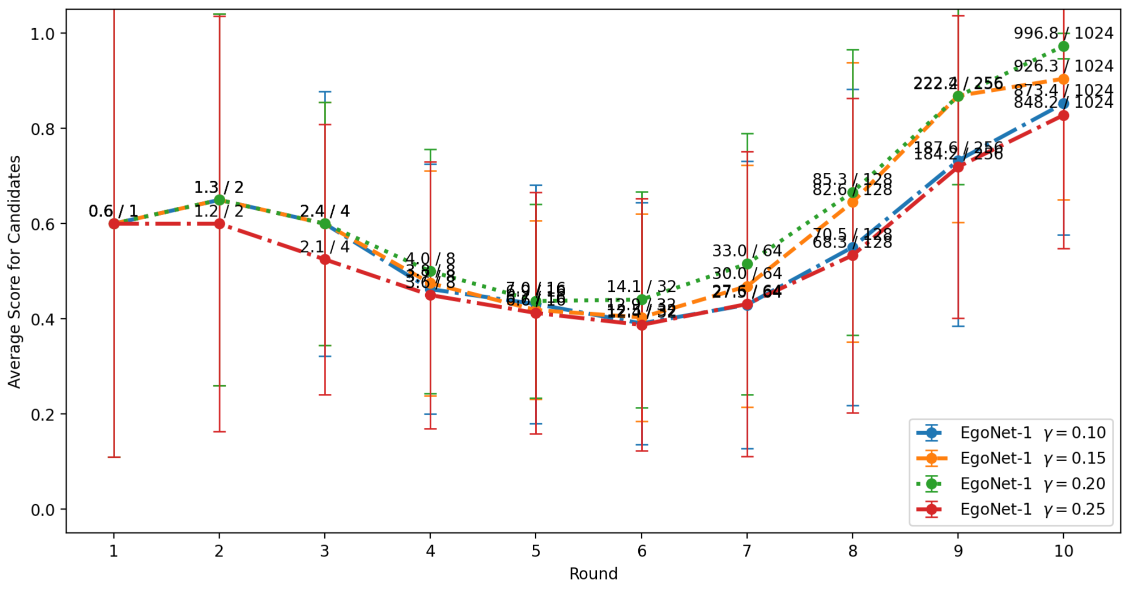


Figure 4.10: Average framework round by round score with standard deviation for the 10 graph instances generated by the BA model with $n = 1024$ and $\rho = 0.2$. Framework execution using edges in EgoNet-1 and comparing values for the weighting parameter γ.

4.3.2 Matching Result

In Figure 4.11 and 4.12 we have the comparison of the final matching result for the framework and for the SANA algorithm for $\rho = 0.1$ and $\rho = 0.2$, respectively.

For instances with $\rho = 0.1$, i.e., a non-aggressive edge sampling process, the framework's performance, regardless of the γ parameter, achieved 100% accuracy. In the evaluation for the BA model, unlike the $G(n, p)$ model, the SANA algorithm exhibited an average accuracy of 70%. Analyzing individual instances, we observe extreme behavior: SANA performed either with low accuracy (below 1%, with a maximum of 8 correct matches out of the 1024 possible pairings) or exceptionally high (correctly identifying all pairs except for instance 10, where there were 4 mismatches).

For graphs with $\rho = 0.2$, the results of the framework were notably higher for the BA model than for the $G(n, p)$ model. On average, using edges in EgoNet-1 as the sorting feature and $\gamma = 0.20$, the accuracy was 97%, with accuracies for other γ values ranging between 83% and 90%. When evaluating individual instances, the framework almost always presented a score above 85%, but for some instances it showed considerably lower performance for specific γ values. In instance 4, the accuracy was lower for $\gamma = 0.10$ and $\gamma = 0.25$, in instance 7 for $\gamma = 0.25$, and in instance 8 for $\gamma = 0.10$ and $\gamma = 0.15$. The SANA algorithm exhibited the same extreme behavior, achieving high performance in only 2 out of 10 instances, resulting in an average accuracy of 21%.

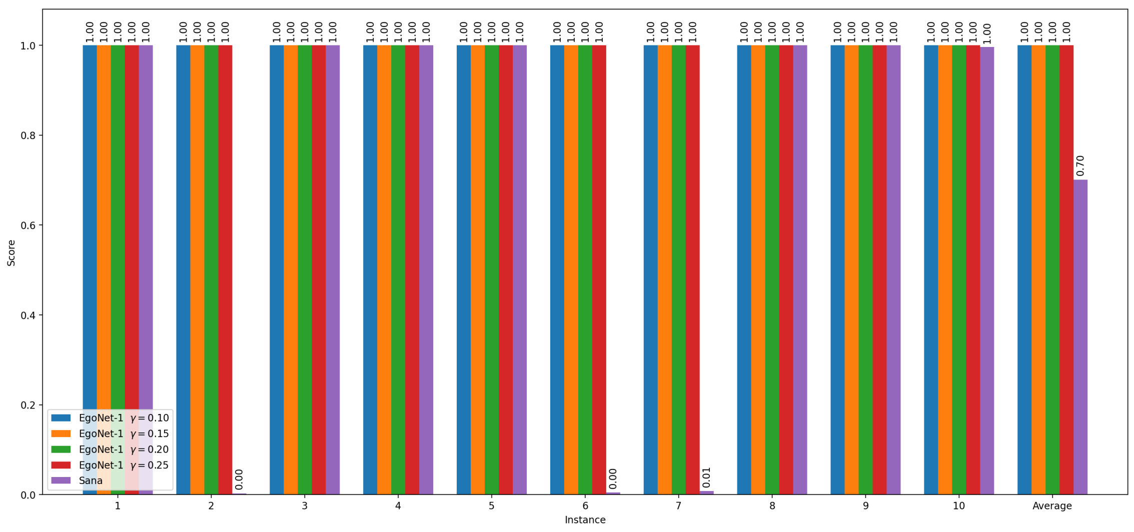


Figure 4.11: Performance comparison for each instance and on average for different parameterizations of the framework and the SANA algorithm. Executions under the 10 instances generated by the BA model using $n = 1024$ and $\rho = 0.1$.

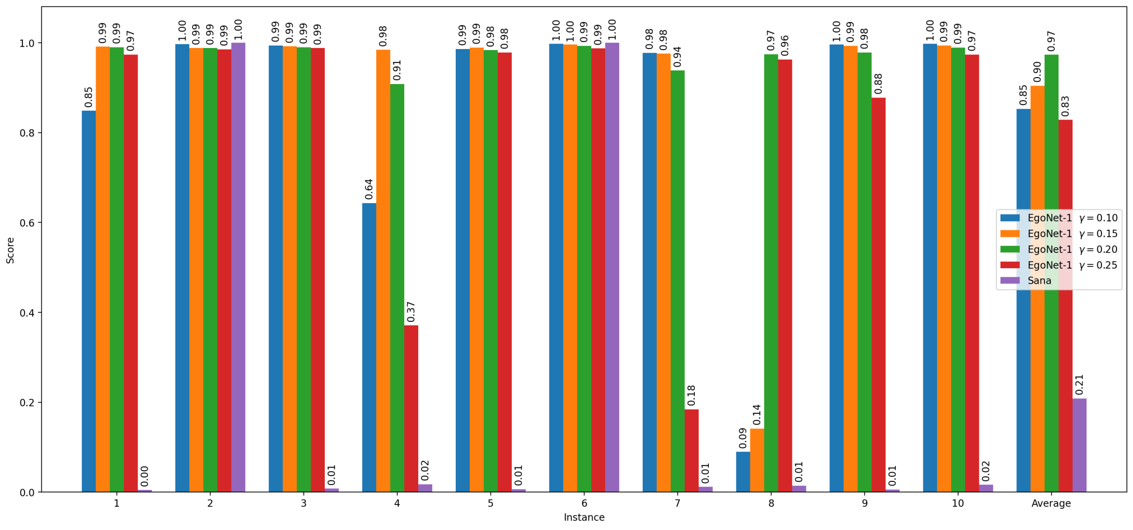


Figure 4.12: Performance comparison for each instance and on average for different parameterizations of the framework and the SANA algorithm. Executions under the 10 instances generated by the BA model using $n = 1024$ and $\rho = 0.2$.

4.4 Differences in results between Erdős-Rényi and Barabási-Albert

Comparing the results obtained from graph instances using the Erdős-Rényi model ($G(n,p)$) and the Barabási-Albert model (BA), it is evident that in most cases, the performance for instances using the BA model was superior. This is particularly noticeable for the SANA algorithm, used for comparison, as its performance in the $G(n,p)$ model was consistently close to zero, regardless of the removing edge probability ρ. For the framework of this dissertation, the differences become evident when $\rho = 0.2$, where the performance is at least 25% higher on average.

These results indicate that graph matching algorithms not only depend on the network topology but can also be sensitive to the characteristics of that topology. In this particular case, there are significant differences between the topologies created by each of these models. For instance, in BA graphs, the degree distribution follows a power-law, meaning they have a few highly connected nodes (hubs). In contrast, the $G(n,p)$ model has a binomial distribution concentrated around the average degree. Another feature is the clustering coefficient, where BA graphs tend to have higher clustering coefficients compared to $G(n,p)$ graphs. These characteristics, especially the presence of hubs in BA graphs, can make them more robust to noise and perturbations. As a consequence, even with missing or incorrect connections, the overall structure of BA graphs can remain relatively preserved. This may indicate the reason why, even for $\rho = 0.2$, the framework was able to achieve superior results for the BA model.

Chapter 5

Conclusions

This dissertation introduces a novel seedless framework based on Personalized PageRank fingerprints for the graph matching problem. The main contribution of this work is an addition to other works in the context of graph matching, especially for systems that do not rely on a seed set of pre-mapped node pairs. The newly defined fingerprints and distance metrics using Personalized PageRank represent a significant advancement over existing approaches based on Personalized PageRank embeddings, which did not account for anchor nodes.

The evaluations were conducted using pairs of similar graphs based on synthetic generative models. In this process, an original graph is generated using the desired model, and pairs of similar graphs are independently generated by subjecting the original graph to an edge sampling process that uniformly removes edges from the original graph with a given input probability. Applying this process allowed assessing the framework's resilience to noise between the compared graphs.

This study also evaluated the proposed framework under different parameters for candidate sorting features and final distance weighting. The selected parameters for assessing performance per instance and on average were the candidate sorting feature as edge in EgoNet-1 and a final weighting γ between 0.10 and 0.25 (corresponding to the metric's contribution without the anchors). These preliminary parameter settings were the most successful, but there is potential for tuning these and other parameters to enhance the final framework performance.

The obtained results outperformed the algorithm SANA used for comparison, achieving accuracy values of up to 97% on averaged across instances with a 20% probability of edge removal for the Barabási–Albert (BA) model. Results obtained with a 10% edge removal probability showed 100% accuracy, regardless of the tested model or vertex candidate ordering. For a 20% edge removal probability, accuracy averaged between 39% and 58% for the $G(n, p)$ model instances and averaged between 83% and 97% for the BA model instances. These results highlights the effectiveness of Personalized PageRank fingerprints in capturing node roles within the network structure.

Independent of the obtained results, it is important to note that the probability of edge removal applied to the edge sampling process significantly affects the outcomes depending on the evaluated instance. This suggests that despite having some resilience to noise introduced by edge sampling, both the instances and the applied generative model also impact how the framework will perform. In addition, this is also related to the interference that noise can cause to the topological structure of nodes in the network. For an edge removal probability of 20%, this implies a 64% probability of the same edge being present in both graphs of the input pair of similar graphs, according to the employed edge sampling process.

5.1 Future Work

The work in this dissertation and the results obtained provide a foundation for further research to better evaluate and enhance the framework, as outlined below.

Parameter tuning

This work investigated the variation of the candidate sorting feature and the final weighting parameter. Nonetheless, there remains significant potential to explore other variations, including parameters that were held constant, such as those related to the PPR execution. Conducting an analysis could provide more insights into how parameter changes impact the performance and accuracy of the graph matching framework. This analysis could help identify more efficient and robust parameters.

Evaluation on real networks

This work evaluates the use of the framework on synthetic networks, constructed using two generative models. Future work may include assessing the framework's performance on real-world networks from different domains, such as social networks and biological networks. This could provide a better understanding of the framework's applicability and effectiveness in diverse scenarios.

Comparison with other seedless graph matching algorithms

This work compared the performance of the framework with the existing SANA [10] algorithm. For future work, a comparative analysis with other seedless graph matching algorithms, such as REGAL [8], can highlight the framework's strengths and weaknesses. This comparison could provide valuable insights into areas where it excels or can be improved.

References

[1] NEWMAN, M. *Networks*. 2 ed. London, England, Oxford University Press, jul. 2018.

[2] BARABASI, A.-L. *Linked: How Everything Is Connected to Everything Else and What It Means for Business, Science, and Everyday Life*. Plume Books, abr. 2003.

[3] NARAYANAN, A., SHMATIKOV, V. "De-anonymizing Social Networks". In: *2009 30th IEEE Symposium on Security and Privacy*, pp. 173–187, 2009. doi: 10.1109/SP.2009.22.

[4] GUZZI, P. H., MILENKOVIĆ, T. "Survey of local and global biological network alignment: the need to reconcile the two sides of the same coin", *Briefings in Bioinformatics*, v. 19, n. 3, pp. 472–481, 01 2017. ISSN: 1477-4054. doi: 10.1093/bib/bbw132. Available in: <https://doi.org/10.1093/bib/bbw132>.

[5] CAI, H., ZHENG, V. W., CHANG, K. C.-C. "A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications", *IEEE Transactions on Knowledge and Data Engineering*, v. 30, n. 9, pp. 1616–1637, 2018. doi: 10.1109/TKDE.2018.2807452.

[6] PEDARSANI, P., FIGUEIREDO, D. R., GROSSGLAUSER, M. "A Bayesian method for matching two similar graphs without seeds". In: *2013 51st Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1598–1607, 2013. doi: 10.1109/Allerton.2013.6736720.

[7] YARTSEVA, L., GROSSGLAUSER, M. "On the performance of percolation graph matching". In: *Proceedings of the First ACM Conference on Online Social Networks*, COSN '13, p. 119–130, New York, NY, USA, 2013. Association for Computing Machinery. ISBN: 9781450320849. doi: 10.1145/2512938.2512952. Available in: <https://doi.org/10.1145/2512938.2512952>.

[8] HEIMANN, M., SHEN, H., SAFAVI, T., et al. "REGAL: Representation Learning-based Graph Alignment". In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, p. 117–126, New York, NY, USA, 2018. Association for Computing Machinery. ISBN: 9781450360142. doi: 10.1145/3269206.3271788. Available in: <`https://doi.org/10.1145/3269206.3271788`>.

[9] SAB, G. A. A. *Building Deterministic Node Representations from Personalized PageRank Sequences.* M.Sc. dissertation, UFRJ/COPPE, Rio de Janeiro, Rio de Janeiro, BR, 2021.

[10] MAMANO, N., HAYES, W. B. "SANA: simulated annealing far outperforms many other search algorithms for biological network alignment", *Bioinformatics*, v. 33, n. 14, pp. 2156–2164, 02 2017. ISSN: 1367-4803. doi: 10.1093/bioinformatics/btx090. Available in: <`https://doi.org/10.1093/bioinformatics/btx090`>.

[11] TABAK, P. *Distance Based Canonical Labeling Algorithms with Applications to Graph Matching.* M.Sc. dissertation, UFRJ/COPPE, Rio de Janeiro, Rio de Janeiro, BR, 2020.

[12] MCKAY, B. D., PIPERNO, A. "Practical graph isomorphism, II", *Journal of Symbolic Computation*, v. 60, pp. 94–112, 2014. ISSN: 0747-7171. doi: https://doi.org/10.1016/j.jsc.2013.09.003. Available in: <`https://www.sciencedirect.com/science/article/pii/S0747717113001193`>.

[13] KAZEMI, E., HASSANI, S. H., GROSSGLAUSER, M. "Growing a graph matching from a handful of seeds", *Proc. VLDB Endow.*, v. 8, n. 10, pp. 1010–1021, jun 2015. ISSN: 2150-8097. doi: 10.14778/2794367.2794371. Available in: <`https://doi.org/10.14778/2794367.2794371`>.

[14] BRIN, S., PAGE, L. "The anatomy of a large-scale hypertextual Web search engine", *Computer Networks and ISDN Systems*, v. 30, n. 1, pp. 107–117, 1998. ISSN: 0169-7552. doi: https://doi.org/10.1016/S0169-7552(98)00110-X. Available in: <`https://www.sciencedirect.com/science/article/pii/S016975529800110X`>. Proceedings of the Seventh International World Wide Web Conference.

[15] PAGE, L., BRIN, S., MOTWANI, R., et al. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66, Stanford InfoLab, November 1999. Available in: <`http://ilpubs.stanford.edu:8090/422/`>. Previous number = SIDL-WP-1999-0120.

[16] GLEICH, D. F. "PageRank Beyond the Web", *SIAM Review*, v. 57, n. 3, pp. 321–363, 2015. doi: 10.1137/140976649. Available in: <`https://doi.org/10.1137/140976649`>.

[17] GONÇALVES DAMASCENO, R. "Personalized PageRank Fingerprint Framework for Seedless Graph Matching". 2024. Available in: <`https://github.com/damascenorafael/graph-matching-ppr-framework`>. Accessed: August 1, 2024.

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

[19] BARABÁSI, A.-L., ALBERT, R. "Emergence of Scaling in Random Networks", *Science*, v. 286, n. 5439, pp. 509–512, 1999. doi: 10.1126/science.286.5439.509. Available in: <`https://www.science.org/doi/abs/10.1126/science.286.5439.509`>.

[20] HAYES, W. B. "Simulating Annealing Network Aligner". 2017. Available in: <`https://github.com/waynebhayes/SANA`>. Accessed: November 23, 2022.

Appendix A

Results for Erdős-Rény with $n = 512$

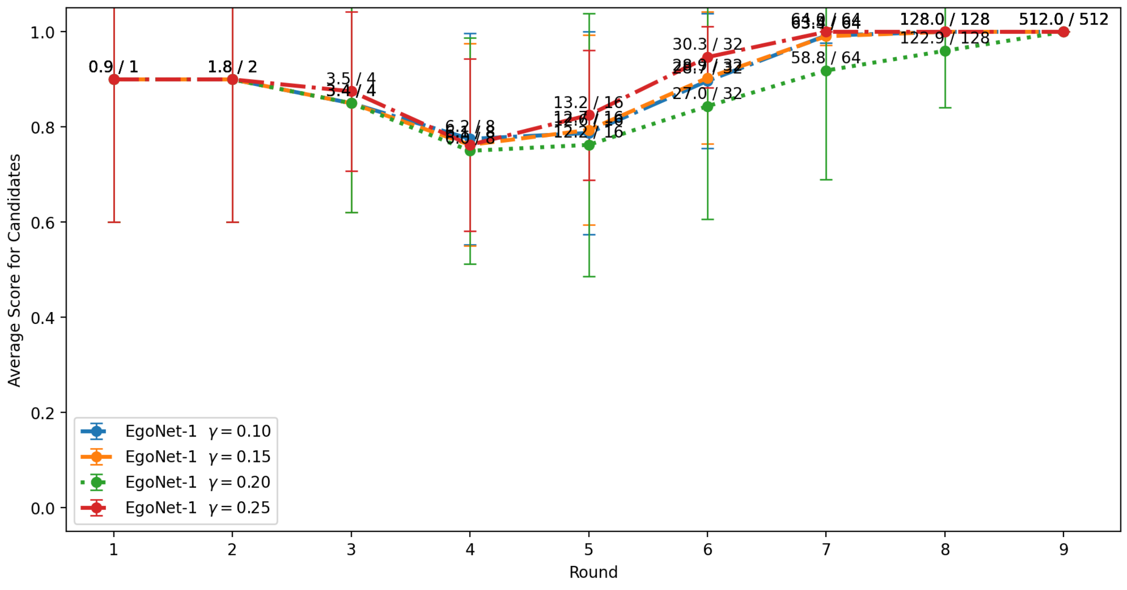


Figure A.1: Average framework round by round score with standard deviation for the 10 graph instances generated by the $G(n, p)$ model with $n = 512$ and $\rho = 0.1$. Framework execution using edges in EgoNet-1 and comparing values for the weighting parameter γ.

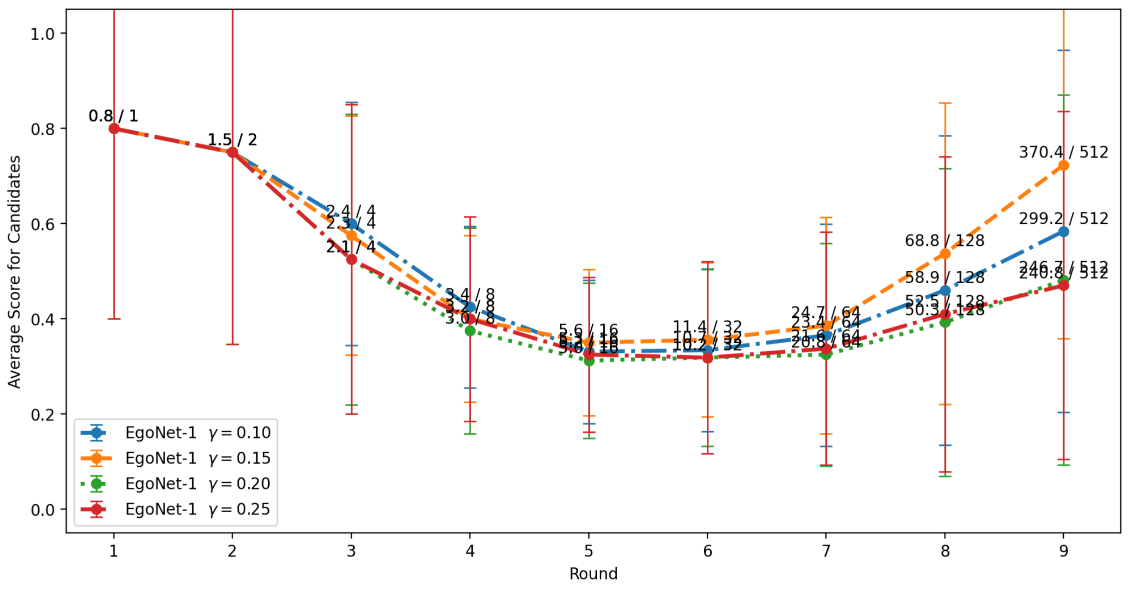


Figure A.2: Average framework round by round score with standard deviation for the 10 graph instances generated by the $G(n, p)$ model with $n = 512$ and $\rho = 0.2$. Framework execution using edges in EgoNet-1 and comparing values for the weighting parameter γ.

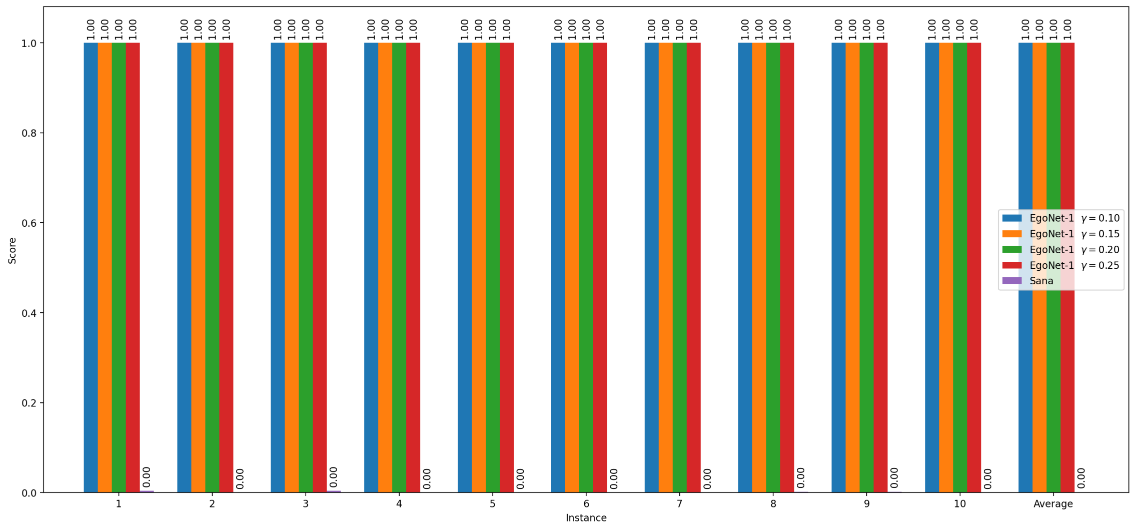


Figure A.3: Performance comparison for each instance and on average for different parameterizations of the framework and the SANA algorithm. Executions under the 10 instances generated by the $G(n, p)$ model using $n = 512$ and $\rho = 0.1$.

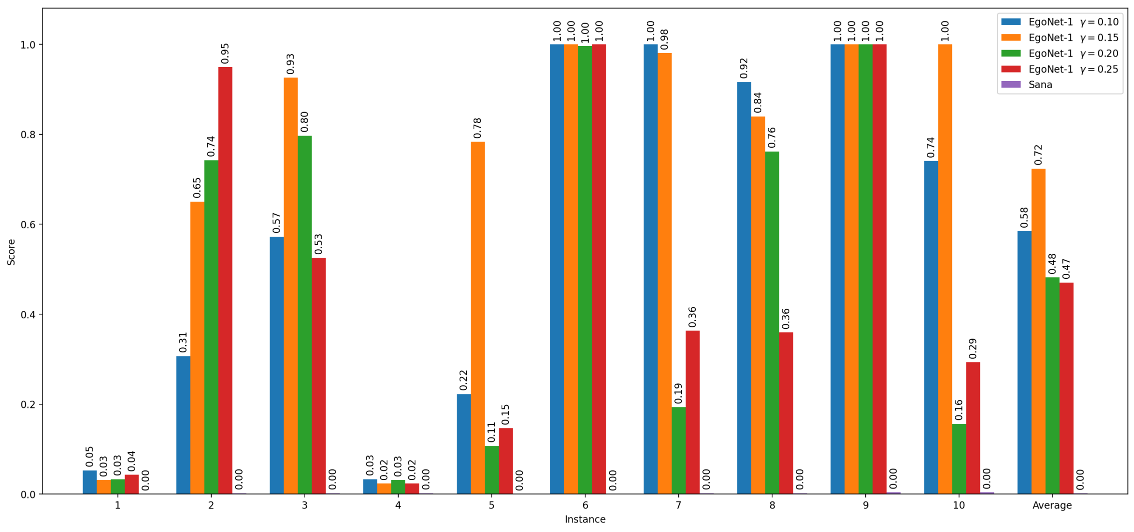


Figure A.4: Performance comparison for each instance and on average for different parameterizations of the framework and the SANA algorithm. Executions under the 10 instances generated by the $G(n, p)$ model using $n = 512$ and $\rho = 0.2$.

Appendix B

Results for Barabási–Albert with $n = 512$

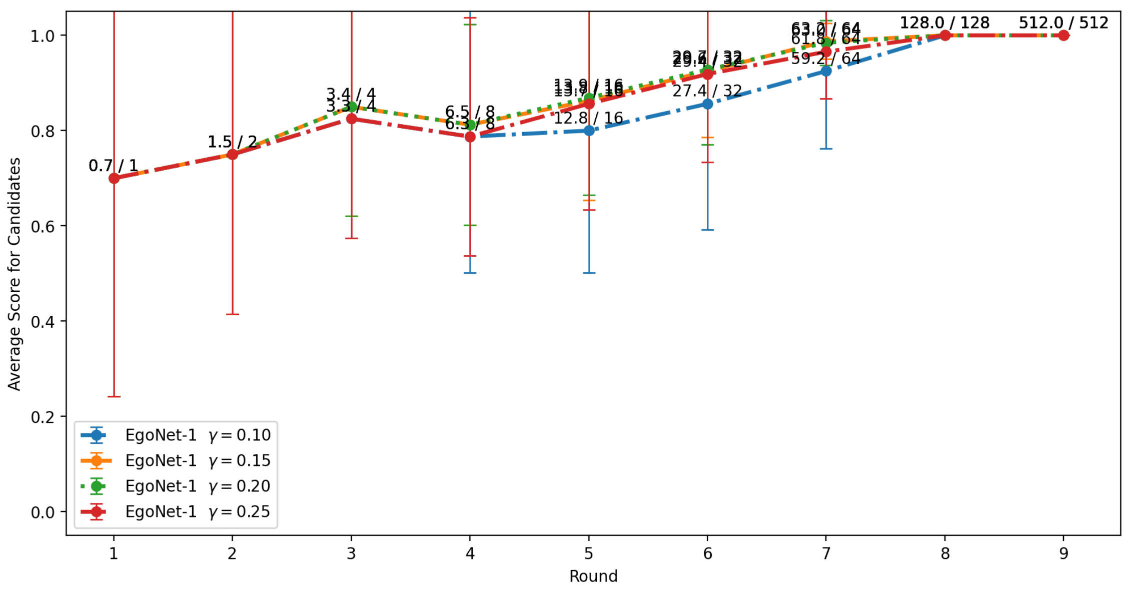


Figure B.1: Average framework round by round score with standard deviation for the 10 graph instances generated by the BA model with $n = 512$ and $\rho = 0.1$. Framework execution using edges in EgoNet-1 and comparing values for the weighting parameter γ.

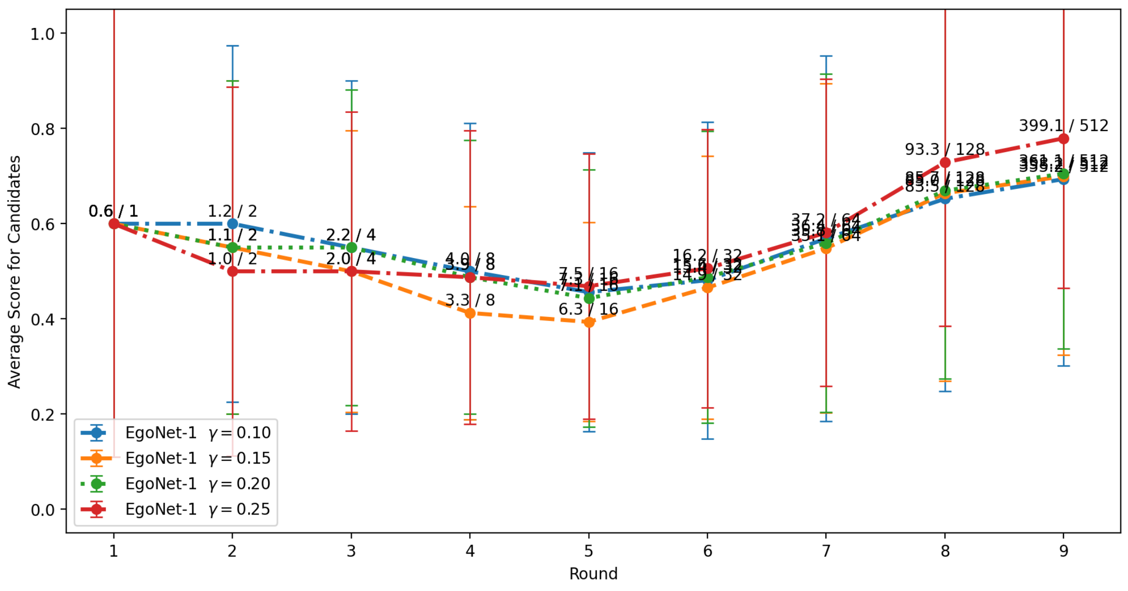


Figure B.2: Average framework round by round score with standard deviation for the 10 graph instances generated by the BA model with $n = 512$ and $\rho = 0.2$. Framework execution using edges in EgoNet-1 and comparing values for the weighting parameter γ.

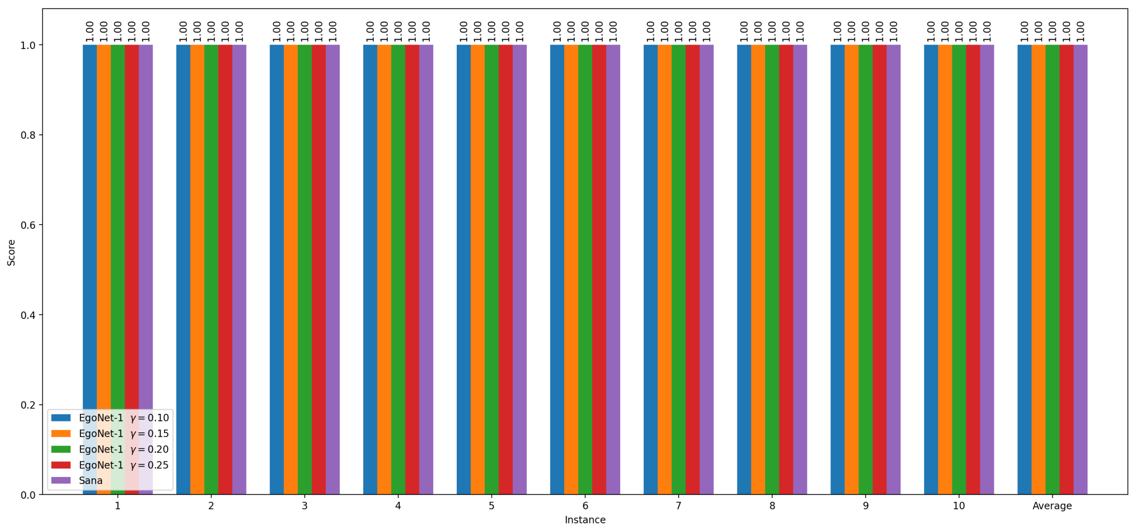


Figure B.3: Performance comparison for each instance and on average for different parameterizations of the framework and the SANA algorithm. Executions under the 10 instances generated by the BA model using $n = 512$ and $\rho = 0.1$.

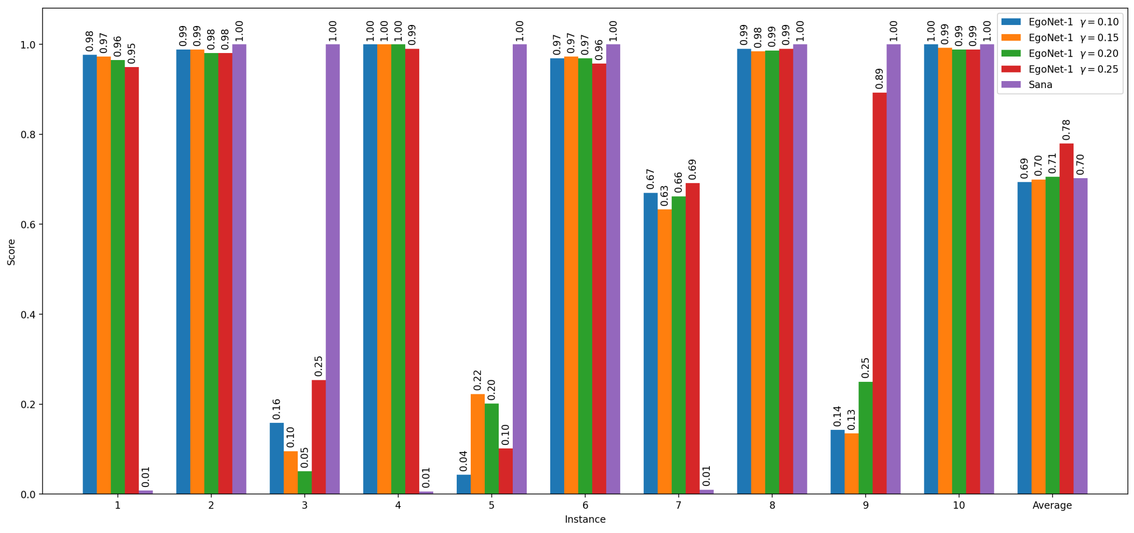


Figure B.4: Performance comparison for each instance and on average for different parameterizations of the framework and the SANA algorithm. Executions under the 10 instances generated by the BA model using $n = 512$ and $\rho = 0.2$.