Similarity in Temporal Graphs

A Thesis

submitted to the designated
by the General Assembly of Special Composition
of the Department of Computer Science and Engineering
Examination Committee

by

Antonis Koursoumis

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

WITH SPECIALIZATION
IN SOFTWARE

University of Ioannina
February 2017

Examining Committee:

- **Evaggelia Pitoura**, Professor, Department of Computer Science and Engineering, University of Ioannina (Advisor)
- **Panayiotis Tsaparas**, Associate Professor, Department of Computer Science and Engineering, University of Ioannina
- **Nikos Mamoulis**, Associate Professor, Department of Computer Science and Engineering, University of Ioannina

Table of Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Antonis Koursoumis, M.Sc. in Computer Science, Department of Computer Science and Engineering, University of Ioannina, Greece, February 2017.
Similarity in Temporal Graphs.
Advisor: Evaggelia Pitoura, Professor.

In all modern networks, calculating metrics and ranking nodes according to their inherent characteristics is a vital part of their analysis. It helps by providing information of the structure of the network, the association and importance of nodes, as well as presenting characteristics and information of the network itself. The main problem lies in redefining already existent models and algorithms that are built for static networks to also apply their calculations, effectively, on networks that evolve over time. This must not be done by completely excluding node structure or other information they already include but by extending them to also calculate temporal metrics. Such a change is essential due to the amount of data being extracted everyday whose components include rich temporal information. This work focuses on altering distance, diameter and centrality metrics as well as the SimRank algorithm to fit temporal data extracted from two different datasets restructured in three different graph implementations; an undirected, a directed and a bipartite one. The resulting definitions, models and implementations are analyzed and ranking algorithms were used to evaluate their significance.

Εκτεταμενη Περιληψη

Αντώνης Κουρσούμης, Μ.Δ.Ε. στην Πληροφορική, Τμήμα Μηχανικών Η/Υ και Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Φεβρουάριος 2017.
Ομοιότητα σε Εξελισσόμενους Γράφους.
Επιβλέπων: Ευαγγελία Πιτουρά, Καθηγήτρια.

Με την εξέλιξη των αυτόματων εργαλείων συλλογής δεδομένων, η δυνατότητα μας να δημιουργούμε και να αποθηκεύουμε πληροφορία έχει αυξηθεί δραματικά τις τελευταίες δεκαετίες. Μεγάλο μέρος αυτής της πληροφορίας χρησιμοποιεί την δομή γράφων για την οργάνωση των δεδομένων. Οι γράφοι αποτελούν ένα κεντρικό κομμάτι της σύγχρονης εποχής. Υπάρχουν παντού. Πολλοί από αυτούς αλλάζουν συνεχώς με το πέρασμα του χρόνου. Βασικότερα παραδείγματα αυτής της εξέλιξης είναι τα πληροφοριακά δίκτυα του διαδικτύου και τα κοινωνικά δίκτυα που περιέχουν εκατομμύρια ή και δισεκατομμύρια ακμές και κόμβους. Η ανάλυση τέτοιων γράφων, που αντιπροσωπεύουν τέτοιο πλήθος εξελισσόμενης και διασυνδεδεμένης πληροφορίας, βρίσκει πολλαπλές εφαρμογές. Η ανάλυση δομών στα σύγχρονα κοινωνικά δίκτυα, η μελέτη καταναλωτικών προτύπων σε ηλεκτρονικά καταστήματα και η ανάλυση των αλληλεπιδράσεων πρωτεϊνών σε βιοϊατρικά δίκτυα είναι λίγα μόνο παραδείγματα.

Στον τομέα των γράφων, υπάρχει μεγάλο πλήθος αλγορίθμων που κύριο σκοπό έχουν την εξαγωγή μετρικών και την κατάταξη στοιχείων του γράφου ανάλογα με αυτές. Παρόλα αυτά, οι γράφοι αυτοί δεν περιέχουν καμία πληροφορίας σχετικά με την εξέλιξη των στοιχείων τους στην πάροδο του χρόνου. Σε αυτή την εργασία, αναλύονται πολλαπλοί τρόποι μετατροπής τέτοιων αλγορίθμων και δομών, με κύριο στόχο την εφαρμογή τους σε γράφους οι οποίοι εξελίσσονται στον χρόνο. Τέτοιοι γράφοι περιέχουν πληροφορία που δεν μπορεί να εξαχθεί με την χρήση των στατικών αλγορίθμων.

Αρχικά αναλύεται η δυνατότητα μετατροπής ενός από τους κυριότερους αλγορίθ-

μους μέτρησης ομοιότητας κόμβων σε γράφους, του SimRank, προκειμένου να εξάγει την ίδια πληροφορία, αποτελεσματικά, από χρονικά εξελισσόμενους γράφους. Στη συνέχεια μελετώνται, με τον ίδιο στόχο, αλγόριθμοι για την μέτρηση απόστασης, κεντρικότητας και διαμέτρου. Τέλος, παρουσιάζονται τα αποτελέσματα αυτών των μετατροπών και αναλύεται η αποτελεσματικότητά τους μέσω της εφαρμογής τους σε πραγματικά, εξελισσόμενα δεδομένα.

Κύριος στόχος αυτής της εργασίας είναι να παρουσιάσει την σημαντικότητα της μετατροπής υπαρχόντων αλγορίθμων ώστε να δουλεύουν σε εξελισσόμενα δεδομένα καθώς και της αποτελεσματικότητα μίας τέτοιας μετατροπής.

Chapter 1

Introduction

1.1 Motivation

Social networks, communication network analysis, or road networks are only three examples out of the field of graphs that change in time. Graphs that change in time are graphs that receive or lose nodes or edges or have their attributes changed over a time period. A lot of research in this field can be found that defines temporal graphs [1] [2] [3]. Recent research focuses on the evolution of traditional graph algorithms and metrics to work on the temporal versions of graphs. Despite the amount of the work conducted, there are a lot of algorithms and measures that are not yet converted, to work on a temporal basis and whose implementation and analysis poses an interesting challenge. If their modification to fit on temporal data is successfull, they can enable a deeper analysis and understanding of modern networks. Based on this, this work implements multiple metrics and ranking algorithms to run on temporal graphs and analyses the results.

1.2 Problem Statement

As of today, there exists no temporal representation of SimRank, one of the most prevalent algorithms for similarity measurement in graphs. In addition, work has been done towards modifying traditional graph measurements such as centrality, distance and diameter to work on evolving graphs [4] [5] but there has been no implementation of the resulting temporal formulas and thus, no evaluation either. Such changes can enhance the traditional static algorithms to work on their evolving networks and reveal underlying information that would otherwise be inaccessible. The change of static algorithms to work on evolving networks is not limited on the algorithms listed here. The methodology required to achieve such changes can also be analyzed and experimented on to figure if it can fit more algorithms and definitions.

1.3 Contribution

This work focuses on modifying the SimRank definition so that it works fast, efficiently and in-memory on different temporal graphs. Such graphs are bipartite and undirected collaboration networks and a directed citation network. In addition, centrality, distance and diameter temporal formulas are defined and implemented in order to measure the same graphs. Multiple versions of SimRank are analyzed to find out which one is best suited for which set of data and find sufficient definitions and algorithms. Apart from the apparent contribution of new algorithms and definitions that fit on evolving graphs, the methodology used to modify said algorithms can also be analysed and extended to fit different algorithms and definitions. Finally, all resulting measures and ranks were analysed and evaluated using multiple rankging algorithms proposed throughout this work.

1.4 Outline

Chapter 2 contains a survey of the related work. Chapter 3 introduces all definitions and theoretical solutions as well as the corresponding data models and algorithms used. Chapter 4 evaluates the results of the implementation. Finally, Chapter 5 concludes the work and suggests possible future research.

Chapter 2

Related Work

2.1 Graphs

On the field of graph databases, the need for handling of enormous datasets led to a research spree in previous years that created both a solid data model basis as well as multiple commercial implementations of systems that support those models. Angles and Gutierrez present the work that has been conducted in the area of graph database modeling, concentrating on data structures, query languages and integrity constraints [6]. Jouili and Vansteenberghe present a distributed graph database comparison framework that analyses and compares the main commercial implementations on the field [7]. Based on graph theory that preceded these models and engines, multiple works were published around calculating measures based on them. Basic graph measures such as centrality, diameter, pair distance and others were analysed. Agarwal and Mahata [8] analysed social graphs to better understand ties between nodes and find ways to promote them. Based on the field of graphs and the modeling of the web as such, Page, Brin, Motwani and Winograd developed PageRank [9], a ranking

algorithm for web and citation networks, who then was used by Google as the core algorithm of its search engine. In addition, and of great importance to this work, Jeh and Widom developed SimRank [10], an algorithm that calculates similarity based on the structure of a network. Rothe et al. expanded SimRank and Personalized PageRank to CoSimRank [11] by creating equivalent formalizations that are faster or more accurate. In addition, Fogaras and Rácz introduced algorithms [12] to compute such metrics on a bigger scale. Sun and Han went ahead and listed [13] most of the ranking algorithms proposed over the years. Finally, many more graph measures and metrics were proposed and analysed over the years creating numerous new fields around graph databases.

2.2 Temporal and Evolving Graphs

However, the evolution of technologies led to the evolution of the graphs and given the seer size of the data stored and processed every day, more information became available. The main attribute that could be further analysed was time and so, temporal graphs were defined. Kostakos [1] introduced the idea of temporal graphs as a representation that encodes temporal data into graphs while retaining the temporal information of the original data. He also included a number of metrics that can be used to study such graphs. Temporal graphs are split in two distinct categories: temporal and evolving [2]. The former include lifetime information for nodes and edges and the latter handle time as consecutive snapshots of the graph where a node or edge exists if it is present on the corresponding snapshot. Some, went ahead to use such evolving networks [14] to analyse the main properties and how they connect to the evolution and underlying structure. They found that in the case of evolving graphs, given specific network structure, the evolution of the network can sometimes be dominated by effects solely linked to structural properties. This analysis was part of the motivation of this work given that the resulting evaluation showed that structural information can be as important as temporal in the case of evolving graphs. This is the main reason that SimRank was chosen to implement both the structural and temporal similarity rankings in the same measure.

2.3 Temporal Measures

At the same time that temporal and evolving graphs emerged, metrics started being defined for them. Mainly metrics that were already present for static graphs but would help analyse the temporal versions as well. Rozenshtein and Gionis defined and implemented PageRank for temporal graphs [15] where activity is represented by time sequences. On the other hand, Mariani and Medo [16] as well as Bahmani and Kumar [17] revealed flaws in the mix of PageRank and temporal data and suggested ways to solve such issues. The former experimented on real data and resulted that the static PageRank approach is inappropriate for many systems and suggested that time-dependent algorithms based on temporal linking patterns are used instead to rank the nodes. Kumar et al tackled the issue of computing PageRank on temporal graphs and suggested a stylized model and algorithm of graph crawling to counter the naive method of the original PageRank. Some of the changes suggested in these works are also included in this analysis as part of the changes applied on the SimRank algorithm. Nicosia et al. [4] redefined various terms and metrics to fit time-varying graphs and proposed multiple extensions in that direction. Santoro et al. [5] also defined multiple temporal and atemporal indicators for both evolving and temporal graphs. Part of their analysis was also implemented in this work in order to more efficiently modify the measuring algorithms and their corresponding definitions. In specific, their use of journeys instead of traditional graph paths lied at the core of the change suggested in this work. Kossinets and Watts [18] defined and analyzed homophily on an evolving social network. Tong, Papadimitriou, Faloutsos and Yu used [19] many of these measures to monitor and try to corellate them with node proximity in an evolving network and finally, on the modeling part of the field, Campos, Mozzino and Vaisman, recently [3] proposed a new model and query language to handle attribute graphs on a real GDB.

2.4 Summary

Although many have defined temporal and evolving graphs and metrics for them, algorithms remain that do not have temporal equivalents. Changes can be made to define new metrics and help by extending the analysis of temporal and evolving graphs. The flaws discovered in ranking algorithms [16] [17] highlight and enhance

the necessity to make changes as the ones used in this work to fit evolving data. Metrics defined and described by Nicosia and Santoro [4] [5] also help create a good basis for the evaluation of our own models and algorithms. Finally, all others [15] [18] [19] helped us figure out the ways to handle algorithms and metrics and how they can be tuned correctly to fit temporal data as well as evaluate our results. This work focuses on fixing absences of the field and hopes to help by providing definitions, as well as experimental results towards that goal.

Chapter 3

Models and Algorithms

3.1 Graphs

3.1.1 Temporal Graphs

Typically, graph structures are used to represent relationships between entities such as individuals or organisations. These relationships are not solely instantiated in a single point in time. On the contrary, in most graph networks, there is temporal information. Recent digital and communication technologies enabled us to capture an unprecedented scale of data about many aspects of human behaviour, such that its temporal richness is adequately preserved. Processing these data in order to model them and create a graph structure that successfully expresses both their static and temporal attributes enabled us to define temporal graphs. Temporal graphs are a graph structured modeled to include temporal information about all its elements.

The most usual representation used in this work, is nodes and/or edges of the graph including a list of their $active$ temporal periods. A temporal graph is a graph in which connections between vertices are active at specific times. Let $G = (V, E)$ be a temporal graph, where V is the set of vertices of G and E is the set of edges of G. An edge $e \in E$ is a triplet (u, v, t, λ) where $u, v \in V, t$ is the starting time, λ is the traversal time to go from u to v starting at time t, and $t + \lambda$ is the ending time. Such a graph denotes temporal information that let us discover new knowledge and patterns not available in non-temporal graphs.

3.1.2 Evolving Graphs

Evolving graphs on the other hand, consist of snapshots, corresponding to instances of the graph that can be analysed independently to extract knowledge. It is defined as a sequence of consecutive graph snapshots $eG = (G_{ts}, ..., G_{te}$ that have the same set of vertices V, but different sets of edges E_t, where $t = ts, ..., te$ the temporal range that the graph represents. The set of edges of each graph snapshot G_t includes all edges that exist in time t.

3.2 Similarity

Similarity is defined as a real-valued function that measures the distance of two objects on a chosen dimension or set of dimensions. Usually a similarity value is higher if the objects are similar and lower otherwise.

SimRank is a similarity measure, applicable in any domain with object-to-object relationships, that rates similarity based on the relationship of objects with other objects. Effectively, it is a measure which considers two objects to be more similar the more they are connected to similar objects. If we denote similarity between objects as $s(a, b) \in [0, 1]$ then in the case of $a = b, s(a, b)$ is defined to be 1. In any other case,

$$s(a,b) = \frac{C}{|I(a)||I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s(I_i(a), I_j(b))$$

where C is a constant between zero and one denoting the uncertainty of the calculation (a structural decay) and $|I(a)|$ and $|I(b)|$ are the number of in-neighbours of nodes a and b respectively. Finally, $I_i(a)$ and $I_j(b)$ are the respective in-neighbors of nodes a

and b.

The main issue with calculating SimRank, on both static and temporal graphs, is the complexity of the operation. To counter this, in their original paper[10], Jeh and Widom proposed some pruning methods to speed up the calculation and reduce the size required to store the graphs. Given that this work calculates and stores everything in memory, such techniques are mandatory and more had to be developed to render the calculation possible in terms of time and space complexity.

The basic SimRank algorithm, in contrast to the definition of SimRank, doesn't use a recursive function but a converging sequence of finite iterations. In practice this means that for every pair of nodes, we only use the close neighbors (one-hop and two-hop) of each node to calculate the similarity score, which is then stored in an adjacency matrix. Every iteration of the algorithm uses the scores produced in the previous iteration (with the only information at the first iteration being the similarity of every node with itself that is, by definition, equal to 1). This creates an algorithm that does not use any recursion and does not require excess memory to store all possible pair similarities. Jeh and Widom, also proved that using the non-recursive version, the similarity scores converge after, about, five iterations to the values produced by the corresponding recursive function. This further enables us to use it.

As is seen in Algorithm 3.2, we do not create a $n * n$ adjacency matrix to store similarities as in the recursive implementation. Instead, we store a pair similarity only if the pair exists and has a non-zero similarity score. This is also extended by the original paper, where neighbors that are more than two hops away are also ignored due to the convergence of their similarity values, with the main node, to 0. This enables us to skip an $n * n$ calculation altogether and only calculate the SimRank scores of every node with its corresponding one-hop and two-hop neighbors. Thus, two different functions are created to return those neighbors based on the graph that is being analysed. These functions are changed to fit the underlying graph accordingly (directed, undirected or bipartite). Algorithm 3.1 presents the default implementation with no pruning and full recursion that calculates the similarity of two nodes in the graph. Algorithm 3.2 is the implementation used as the basis for our work and uses the pruning methods and non-recursive techniques analysed above to calculate all-pair similarity in the graph.

Algorithm 3.1 Recursive SimRank for Static Graphs

```
1: g ← graph
2: c ← decay
3: if node_a = node_b then
4:     return 1
5: end if
6: neighbors_a = neighbors of node_a
7: neighbors_b = neighbors of node_b
8: if !neighbors_a or !neighbors_b then
9:     return 0
10: end if
11: similarity = 0
12: for n_a in neighbors_a do
13:     for n_b in neighbors_b do
14:         pair_sim = sim(n_a, n_b)
15:         similarity += pair_sim
16:     end for
17: end for
18: return similarity * c / neighbors_a * neighbors_b
```

Algorithm 3.2 SimRank for Static Graphs

```
g ← graph
c ← decay
n ← nodes
mi ← iterations
oldsim ← previous similarity scores
sim ← current similarity scores
for n do
    sim[n][n] = 1
    oldsim[n][n] = 0
end for
for iteration in (0, mi) do
    oldsim = deepcopy of sim
    for node_one in n do
        valid_neighbors = oneHopNeighbors + twoHopNeighbors of node_one
        for node_two in valid_neighbors do
            if node_one == node_two then
                continue
            end if
            similarity_one_two = 0
            for neighbor_node_one in oneHopNeighbors(g, node_one) do
                for neighbor_node_two in oneHopNeighbors(g, node_two) do
                    similarity_one_two += oldsim[neighbor_node_one][neighbor_node_two]
                end for
            end for
            neighbor_combinations = #node_one neighbors * #node_two neighbors
            sim[node_one][node_two] = c * similarity_one_two / neighbor_combinations
        end for
    end for
end for
```

3.2.1 Evolving SimRank

The main goal of this work is to incorporate similarity measurements in a temporal context. To achieve this, several methods have been proposed and analysed. The main method is aggregation over time, with the addition of a temporal decay that decreases the importance of neighbors that exist temporally further from the main node. Extending this definition, we define another solution that also includes future snapshots in the calculation. The core calculation remains the same. However, including future snapshots to the calculation changes the results. More on this in the next chapter.

Average Score and Temporal Decay

In this method, we calculate the SimRank score of a node pair and also, apply a cumulative decay. The decay depends on the temporal distance of the neighbor from the final snapshot of the calculation. Given D the temporal decay factor, t_n the time instance of the final snapshot of the graph and t_a, t_b the time instances that a and b exist in, we define an evolving SimRank calculation by altering the default SimRank definition to

$$s_D(a,b) = \frac{C}{|I(a)||I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s(I_i(a), I_j(b)) D^{t_d}$$

where

$$t_d = t_n - \frac{t_a + t_b}{2}$$

is the average temporal distance of both in-neighbors from the last time instance of the graph. This ensures that the furthest the in-neighbors are, temporally, from the final time instances of the graph the less they contribute to the overall similarity score.

Note here that an evolving graph is a graph that is split into snapshots that correspond to the various time units in the graph. This means that only edges and their nodes, that exist in the corresponding snapshot, exist in the snapshot. However, a step for the preprocessing of the graph, that ensures it is split in snapshots based on time, would be really expensive computationally. To counter this, we modify the one-hop and two-hop neighbors functions to incorporate time and only return nodes that are connected to the main node through an edge that existed at the given time. To implement this, the main algorithm has to be changed so that the SimRank calculation takes place each snapshot for all iterations and after the calculation is complete, hold

a sum of all snapshot scores for each pair. The final sum is then divided by the total number of snapshots to calculate the average. The calculation can be seen in Algorithm 3.3. Algorithm 3.4 shows the final part of the Aggregate SimRank for Evolving graphs where all pair aggregate values are divided by the total number of snapshots to get the final average similarity score. The corresponding one-hop and two-hop neighbors functions are changed so that they only return neighbors on the given snapshot. Note also, that the one-hop and two-hop neighbors functions are built so that they also calculate neighbors for a range of snapshots. This is to make them reusable for all other temporal algorithms as well. These functions also change to fit the different datasets that are used for the experiments.

Algorithm 3.3 Aggregate SimRank for Evolving Graphs

```
g ← graph, n ← nodes of g
c ← decay, d ← temporal decay
mi ← iterations
time_start ← starting time, time_end ← final time
aggregate ← aggregate similarity score
for instance in (time_start, time_end + 1) do
    oldsim ← previous similarity scores
    sim ← current similarity scores
    for iteration in (0, mi) do
        oldsim = deepcopy of sim
        for node_one in n do
            valid_neighbors = oneHopNeighbors + twoHopNeighbors for instance
            for node_two in valid_neighbors do
                if node_one == node_two then
                    continue
                end if
                similarity_one_two = 0
                for neighbor_node_one in oneHopNeighbors(g, node_one, instance) do
                    for neighbor_node_two in oneHopNeighbors(g, node_two, instance) do
                        similarity_one_two += oldsim[neighbor_node_one][neighbor_node_two]
                    end for
                end for
                neighbor_combinations = #node_one neighbors * #node_two neighbors
                sim[node_one][node_two] = c * similarity_one_two / neighbor_combinations
                if iteration == mi - 1 then
                    temporal_decay = d ** (time_end - instance)
                    aggregate[node_one][node_two] += sim[node_one][node_two] * temporal_decay
                end if
            end for
        end for
    end for
end for
```

Algorithm 3.4 Average SimRank score calculation for Evolving Graphs

```
time_start ← starting time
time_end ← final time
aggregate ← aggregate similarity score
for node in aggregate do
    for neighbor in node do
        aggregate[node_one][neighbor] /= |time_end - time_start|
    end for
end for
```

±x Snapshots

The $\pm x$ approach mimics that of Temporal Decay, using a cumulative decay for each snapshot away from the main one. However, only $\pm$ snapshots of the full graph are used to calculate the similarity of node pairs. Similar to the temporal decay algorithm, given D the decay factor, t_n the time of the nth snapshot of the graph and t_a, t_b the snapshot of the graph that a and b connect to their in-neighbors, we define $\pm x$ similarity of nodes a and b as

$$s_{t_n \pm x}(a,b) = \frac{C}{|I_{t_n \pm x}(a)||I_{t_n \pm x}(b)|} \sum_{i=1}^{|I_{t_n \pm x}(a)|} \sum_{j=1}^{|I_{t_n \pm x}(b)|} s(I_i(a), I_j(b)) D^{t_d}$$

where

$$t_d = \frac{(|t_a - t_n|) + (|t_b - t_n|)}{2}$$

is the average distance of both in-neighbors from the nth snapshot of the graph. Similar to the temporal decay method, this ensures that nodes that are temporally closer to the main snapshot contribute more to the similarity score.

To implement this method, the same calculation takes place as in the average scores method but we consider that pair similarity is influenced only by nodes x snapshots away from the given snapshot, not pre-existing it. For example, in the datasets we are using, if we were interested in the similarity of a node pair at year 2000, only nodes that were connected to the given pair at 1998-2002 would be considered given x is equal to 2. The initial algorithm calculates decay based on the absolute value of the difference of the year of the edge so that is also treated successfully using the same algorithm.

3.2.2 Temporal SimRank

In a temporal graph, where all temporal information is available on each edge, we can analyse the graph based on the actual temporal sequence. This means calculating the similarity of nodes based on the similarity of incoming nodes that only existed in previous temporal points. In addition, the traversal time that is required to get from the neighbor to the node under consideration must not exceed the temporal distance of the two. SimRank is proven to be equivalent to the time in which two random walkers are expected to meet, at the same node, if they started at nodes a and b and randomly walked the graph backwards[10]. Based on this, we define the temporal version of this to be equivalent to the time it takes two random walkers to meet, at the same node, if they started at nodes a and b and randomly walked the graph backwards, choosing only nodes that, temporally, pre-existed the node under analysis and whose traversal time does not exceed the temporal distance between the two making sure that a valid temporal sequence is followed. For example, walker W_a arriving to node a from node v at $t(v,a)$ would jump to an incoming neighbor u of node a if the time $t(a,u)$ is lower than $t(v,a) + \lambda(a)$ where $\lambda(a)$ is the time the walker has to wait on node a. In different cases the traversal time λ refers to the edge between two nodes denoting the time it takes to cross that edge and move to another node. The definition remains the same.

Using this method the default definition of SimRank changes to

$$s_{Temp}(a,b,t(u,a),t(v,b)) = \frac{C}{|I_{t_{l_a}}(a)||I_{t_{l_b}}(b)|} \sum_{i=1}^{|I_{t_{l_a}}(a)|} \sum_{j=1}^{|I_{t_{l_b}}(b)|} s(I_i(a), I_j(b), t(a, I_i(a)), t(b, I_j(b)))D_{avg}$$

where

$$t_{l_a} = t, \forall t < t(u,a) + \lambda(a),$$

$$t_{l_b} = t, \forall t < t(v,b) + \lambda(b),$$

the average decay is

$$D_{avg} = \frac{D^{t(u,a)-t(a,I_i(a))} + D^{t(v,b)-t(b,I_j(b))}}{2}$$

and $t(x,y)$ the time of the edge between node x and node y.

Temporal SimRank is defined completely different from its evolving counterpart, in the sense that nodes have to follow a temporal sequence. This means that a node can only be considered connected to nodes that pre-existed it and whose starting time

does not exceed the ending time plus the traversal time of the previous node. This also means that a pairs similarity can only be measured at a specific time for both nodes.

Based on the above definition, the temporal similarity of two nodes depends on the similarity of all their neighbors that, temporally, pre-existed them. Because of this, an approach similar to that of the evolving SimRank is not possible. If we wanted to implement an algorithm that calculated it, we would have to either set a maximum history limit (which limits the available knowledge by removing time instances) or calculate all similarities for every year preceding the one under consideration because every similarity on year x depends on the similarity scores of its neighbors on year $x-1$ (which increases the time and space complexity exponentially). To counter that and because we can still set some structural constraints, the recursive implementation of SimRank can be altered to take time into consideration. This can only be done because there are strict stopping conditions for the recursion. One condition is that (due to the fact that we are moving backwards in time) neighbors that pre-existed will eventually not exist, ending the recursion due to reaching the end of the dataset. The second condition is the same as in all other implementations. It is that neighbors more than two hops away are considered non-similar and are thus, terminating the recursion returning zero as their similarity score. In case we want to be more loose on the definition, we can allow three-hop neighbors to be considered although the complexity increases further. Of course, we are still including a temporal decay factor stating that connections further back in time are less important for the similarity score. The final decay was set equal to the average between each edges specific decay given that their difference from the edge under consideration may vary. For finding the similarity over multiple snapshots, we just average the scores for each snapshot. Finally, it should also be noted that traversal time in our datasets is considered to be at all times equal to one snapshot so it is ommited and all pre-existing nodes are picked instead. Algorithm 3.5 includes all the aforementioned changes.

Algorithm 3.5 SimRank for Temporal Graphs

```
g ← graph
c ← decay, d ← temporal decay
y_a ← snapshot of edge a, y_b ← snapshot of edge b
node_a ← node a, node_b ← node b
hop ← how many hops the algorithm has done
if node_a = node_b then
    return 1
end if
if hop ≥ 2 then
    return 0
end if
neighbors_a = neighbors of node_a where (node_a, neighbor_a).snapshot < y_a
neighbors_b = neighbors of node_b where (node_b, neighbor_b).snapshot < y_b
if !neighbors_a or !neighbors_b then
    return 0
end if
similarity = 0
for n_a in neighbors_a do
    for n_b in neighbors_b do
        edge_a = (node_a, n_a).snapshot
        edge_b = (node_b, n_b).snapshot
        hop += 1
        pair_sim = sim(n_a, n_b, edge_a, edge_b, hop)
        similarity += pair_sim * ((d**(y_a-edge_a) + d**(y_b-edge_b))/2)
    end for
end for
return similarity * c / neighbors_a * neighbors_b
```

3.3 Distance

The distance between two nodes of a graph is the number of edges in a shortest path connecting them. In the case of a directed graph the distance $d(u, v)$ between two

nodes u and v is defined as the length of a shortest path from u to v consisting of directed edges, provided at least one such path exists. So, the problem of calculating distance in an evolving or temporal graph is modified and the shourtest journey is calculated instead between two nodes.

For the shortest path calculation (which is the basic unit for all other metrics) we use algorithm 3.6. It is an implementation of Djikstra for an unweighted graph.

Algorithm 3.6 Basic Shortest Path Implementation using Djikstra Algorithm

```
1: g ← graph
2: source ← source node
3: target ← target node
4: paths = empty dictionary
5: visited = empty dictionary
6: distances = empty dictionary
7: fringe = empty heap
8: fringe.push((0, source))
9: while fringe do
10:    dist, u = fringe.pop
11:    if u in distances then
12:       continue
13:    end if
14:    distances[u] = dist
15:    if u == target then
16:       break
17:    end if
18:    for v in u.neighbors do
19:       uv_dist = distances[u] + 1
20:       if v not in visited or uv_dist < visited[v] then
21:          visited[v] = uv_dist
22:          fringe.push((uv_dist, v))
23:          paths[v] = paths[u] + v
24:       end if
25:    end for
26: end while
```

3.3.1 Evolving Distance

In the case of evolving graphs, the definition of a shortest path remains the same as that of a static graph. We calculate the shortest path of all pairs of nodes of the graph, for each snapshot, and then divide it by the number of snapshots, resulting in the average shortest path length. In addition, we can analyse the change of the shortest path between nodes for each snapshot.

3.3.2 Temporal Distance

In temporal graphs the definition of a path has to change to that of a $journey$. We will define [5] a $journey$ as a temporal extension of the notion of path. More specifically, a journey J is defined as $J = \{(e_1, t_1), (e_2, t_2), ..., (e_k, t_k)\}$, such that $\{e_1, e_2, ..., e_k\}$ is a walk in our graph. J is considered a $journey$ in our graph if and only if $\forall i, 1 \leq i < k, e_i$ exists in t_i and $t_{i+1} > t_i$. Journeys can be thought of as paths over time from a source to a destination and therefore have both a topological and a temporal length. Their topological length is equal to the number of nodes they consist of. The temporal length is the temporal difference between the first and last step of the journey (the time that the first and last edges exist in). All definitions of distance, centrality and diameter are changed to include the definition of journey instead of that of a path.

To implement an algorithm that uses journeys to calculate distance, we only changed the neighbor selection part of the Djikstra algorithm. That part of the algorithm now requires that all neighboring nodes pre-exist the node under analysis. However, marking a node as visited, from an edge that existed at time t, means that we are limiting all paths to pre-exist that specific moment. In the case that the node is visited on time t_1 where $t > t$, then we are excluding possible paths. Figure 3.1 shows an example.

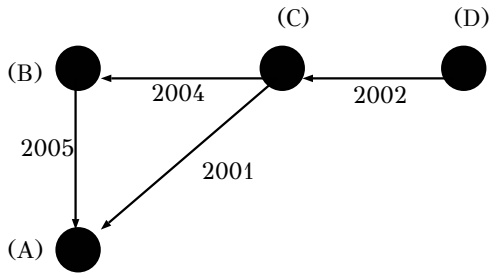


Figure 3.1: In this case, using only the temporal continuity of the paths, node (C) would be analysed first by Djikstra and thus, edge (D)-(C) cannot be included in the path because its time is larger than that of edge (C)-(A). However, in the case of edge (C)-(B), edge (D)-(C) can be included in the path. Based on the default definition though, (C) is first visited from node (A) at time 2001 and is blocking all other nodes from visiting him.

To counter this issue, in the case of algorithm 3.6, despite changing only the condition of neighboring nodes obeying to a temporal continuity, we add an extra conditional statement that checks if the edge that visited an already visited node before, has a lower time unit of the edge under consideration. In the example of Figure 3.1, Djikstra would check node (C) through the edge (B)-(C) that has a larger time variable than edge (C)-(A) and so, it would remove (C) from the visited nodes and reconsider him as a path candidate altogether. The solution is to include the time that our algorithm reaches a the node through an edge. This means that a node is considered visited only in a specific time and if the algorithm reaches it at a time greater than the previous (which means that there might be paths available that were excluded before) the node will be added back to the heap with a different time. Algorithm 3.7 presents the proposed solution.

Algorithm 3.7 Temporal Path Implementation of the Djikstra Algorithm

```
g ← graph
source ← source node
target ← target node
start_time ← start time
end_time ← end time
paths = empty dictionary
visited = empty dictionary
distances = empty dictionary
fringe = empty heap
fringe.push((0, source, start_time))
while fringe do
    dist, u, y = fringe.pop
    if (u, y) in distances then
        continue
    end if
    distances[(u, y)] = dist
    for v in u.neighbors where (v, u).time < y do
        uv_time = (v, u).time
        uv_dist = distances[(u, y)] + 1
        if v in visited but in smaller time or v not in visited or uv_dist < visited[(v,
        uv_time)] then
            visited[(v, uv_time)] = uv_dist
            fringe.push((uv_dist, v, uv_time))
            paths[(v, uv_time)] = paths[(u, y)] + v
        end if
    end for
end while
```

3.4 Centrality

In graph theory and network analysis, indicators of centrality identify the most important vertices within a graph. Centrality concepts were first developed in social

network analysis, and many of the terms used to measure centrality reflect their sociological origin. There are three different measures of centrality. Degree centrality, closeness centrality and betweeness centrality.

Degree centrality [20] is defined as the number of links incident upon a node. It can be easily calculated by measuring the degree of each node. In the case of directed graphs, we define in-degree and out-degree the number of incoming edges and the number of outgoing edges respectively.

$$C_D(u) = deg(u)$$

Closeness centrality is the average length of the shortest path between the node and all other nodes in the graph.

$$C(u) = \frac{\sum_v d(v,u)}{|v|}$$

where v all nodes of the graph despite u and $d(v,u)$ the shortest path between v and u.

Betweeness centrality of a node is equal to the number of times the node acts as a bridge along the shortest path between two other nodes. For each pair of nodes (s,t) in the graph:

$$C_B(u) = \sum_{s \neq u \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(u)$ is the number of those paths that pass through u.

3.4.1 Evolving Centrality

Evolving Degree Centrality, based on the definition of evolving graphs, is equal to the average degree of incoming nodes, on each snapshot of the evolving graph. To implement it in evolving graphs, we only need to average the scores over all the snapshots.

Evolving Closeness Centrality is equal to the average of the average lengths of all the shortest paths between the node and all other nodes, on each snapshot of the evolving graph. In practice, we calculate the closeness centrality score as defined for static graphs for each graph snapshot and then find the average score. To calculate it for an evolving graph, an averaging needs to be conducted as does in the case of

Degree Centrality.
Evolving Betweeness Centrality is equal to the average of the betweeness centrality scores of the node on each consecutive snapshot of the evolving graph. The implementation remains the same as above.

3.4.2 Temporal Centrality

Temporal Degree Centrality of a node u at time t, is equal to

$$Ctemp_D(u,t) = |I_{t-1}(u)| + |O_{t+1}(u)|$$

where $|I_{t-1}(u)|$, $|O_{t+1}(u)|$ are the counts of incoming nodes at time $t-1$ and outgoing nodes at time $t+1$ respectively. To calculate a total we use the average of the individual time scores.
Temporal Closeness Centrality is equal to

$$Ctemp_C(u) = \frac{\sum_v d_j(v,u)}{|v|}$$

where v all nodes of the graph despite u and $d_j(v,u)$ the length of the shortest $journey$ between v and u. To implement it we only need to use the of the temporal Djikstra given in the Temporal Distance subsection
Temporal Betweeness Centrality is equal to

$$Ctemp_B(u) = \sum_{s \neq u \neq t \in V} \frac{\sigma j_{st}(v)}{\sigma j_{st}}$$

where σj_{st} is the total number of shortest journeys from node s to node t and $\sigma j_{st}(u)$ is the number of those journeys that pass through u. Again, the implementation uses the Temporal Djikstra suggested above.

3.5 Diameter

Diameter is defined as the length of the largest shortest path of a graph.

$$diameter = max(SP(G))$$

where $SP(G)$ is a list of the lengths of all shortest paths in graph G.

3.5.1 Evolving Diameter

In the case of evolving graphs, the definition of diameter remains the same as that of a static graph. We calculate the diameter of the graph for each snapshot and then divide it by the number of snapshots, resulting in the average graph diameter. In addition, we can analyse the change of the diameter of the graph for each snapshot. The implementation is just an averaging of the individual snapshot scores.

3.5.2 Temporal Diameter

In temporal graphs the definition of diameter changes slightly so that it becomes equal to the length of the maximum shortest $journey$ of the graph. To implement, we use the Temporal Djikstra algorithm provided above.

3.6 Top-k Measurements and Rankings

To analyse and evaluate our algorithms as well as the datasets they run on, multiple nodes and node pairs measurements and ranking algorithms are used. This helps by providing a more fine-grained analysis of the data. When referring to node pairs, we are referring solely to similarity and distance measures. When referring to single nodes, ranking and all other measures can be used. Note that all the following algorithms are run after the completion of the measuring algorithms analysed above.

3.6.1 Most Similar or Close Over Time Range

This algorithm returns the Top-k pairs that have the highest similarity score over a set range of time. For example, if we want to find the Top-k most similar pairs for a time range, first, a calculation of the similarities or distances would be conducted for each of the given time snapshots and then we would rank them based on the final score.

3.6.2 Drop/Increase Over Time Range

This algorithm corresponds to all measures analysed above. It can be used to rank the drop or increase of pair similarity and distance, single node rank and centrality or

even diameter. To calculate this the difference of the measure between the first and last time unit is used and everything is ranked thereafter. If a more fine-grained approach is required for analysis purposes, multiple smaller time ranges can be queried.

3.6.3 Average Node Similarity or Distance

This algorithm returns the average similarity or distance of a node with its close neighbors (based on the closeness limits set in each experiment). The total sum of the measure with each of its neighbors is calculated and divided by the total number of the neighbors.

3.6.4 Most similar neighbors count

Finally, the most similar neighbors count algorithm returns the number of neighbors (distance limited by the maximum number set in each experiment) that has a higher similarity measure than the threshold set in the experiment. For example, if a node has 10 neighboring nodes with which his similarity is higher than a threshold T, the algorithm will return 10 as the result.

Chapter 4

Evaluation

4.1 Tools

For the creation and handling of the graphs, Python 3.5.2 with the NetworkX 1.10 library was used. NetworkX includes a plethora of static graph algorithms such as distance algorithms and similarity algorithms but none of them was used due to the importance of the base algorithm being tunable in order to work for temporal graphs. Only a few of the variables of these graphs could be changed and thus weren't sufficient for this works requirements. The graph model, management engine and algorithms were all created from scratch. The library was only used to handle the graph units (nodes, edges and attributes on both). In addition multiple Python libraries were used to assist with mathematical calculations and speed some parts of the algorithms up.

For parsing the graphs, only Python was used in the case of the Arxiv Dataset. For the DBLP dataset, the graph readers were written in Java 8. However, the graph was, once again, structured in Python.

4.2 Datasets

In order to measure all metrics on representative datasets and extend their definition to include bipartite, directed and undirected graphs alike, three different implementations were used including data from two different datasets.

4.2.1 Directed Arxiv HEP-PH

Arxiv HEP-PH[21] (high energy physics phenomenology) citation graph is from the e-print arXiv and covers all the citations within a dataset of 34,546 papers with 421,578 edges. When a paper i cites a paper j, the graph contains a directed edge from i to j. If a paper cites, or is cited by, a paper outside the dataset, the graph does not contain any information about this. The papers included in the dataset cover the period from January 1993 to April 2003 (124 months). It begins within a few months of the inception of the arXiv, and this represents essentially the complete history of its HEP-PH section.

Table 4.1: Directed Arxiv HEP-PH Dataset Statistics

Nodes	34546
Edges	421578
Nodes in largest WCC	34401 (0.996)
Edges in largest WCC	421485 (1.000)
Nodes in largest SCC	12711 (0.368)
Edges in largest SCC	139981 (0.332)
Average clustering coefficient	0.2848
Number of triangles	1276868
Fraction of closed triangles	0.05377
Diameter (longest shortest path)	12
90-percentile effective diameter	5

To conduct the experiments, the dataset is, first, loaded in-memory and all nodes are created. Then, all edges are added and for each edge, the source nodes meta data is parsed and processed (there were multiple date template inconsistencies) to include the cite year as an attribute on the corresponding edge.

4.2.2 DBLP

The DBLP[22] computer science bibliography contains the metadata of over 1 million publications, written by over 300 thousand authors in several thousands of journals or conference proceedings series. Although DBLP started with a focus on database systems and logic programming (hence the acronym), it has grown to cover all disciplines of computer science. In practice, it is a collaboration network representing authors as nodes and co-authorship between them as edges connecting them at the time of the publication.

Table 4.2: Undirected DBLP Dataset Statistics

Nodes	317080
Edges	1049866
Nodes in largest WCC	317080 (1.000)
Edges in largest WCC	1049866 (1.000)
Nodes in largest SCC	317080 (1.000)
Edges in largest SCC	1049866 (1.000)
Average clustering coefficient	0.6324
Number of triangles	2224385
Fraction of closed triangles	0.1283
Diameter (longest shortest path)	21
90-percentile effective diameter	8

Bipartite DBLP

To construct a bipartite representation of DBLP, we load the whole dataset in memory and then create a publication node, for every publication listed and an author node, pointing to that publication at the year (year information included as attribute on the edge) it was published, for every author included in the publication. If the author node already exists in the dataset, we just add an extra edge instead of creating a new one. An example can be seen in Figure 4.1.

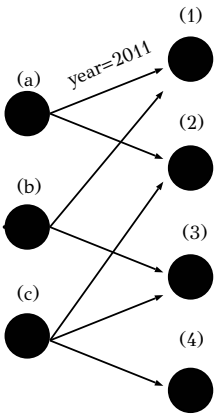


Figure 4.1: Bipartite DBLP example. Nodes (a), (b) and (c) are authors and nodes (1), (2), (3) and (4) are publications. Edges connecting them include the year of publication attribute.

Undirected DBLP

To construct an undirected representation of DBLP, we load the whole dataset in memory and then create one node for each author. For each publication, we create an undirected edge between all author pairs included in the publication with an attribute stating the year the publication was published. An example can be seen in Figure 4.2.

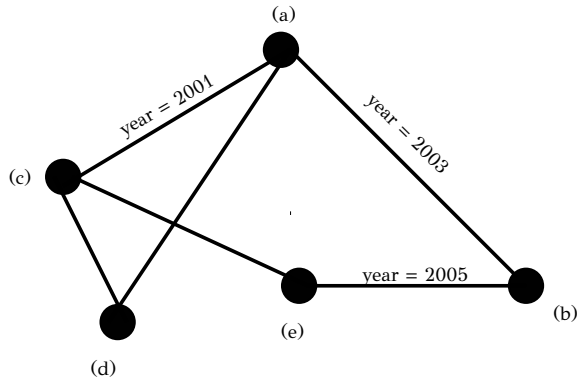


Figure 4.2: Undirected DBLP example. Nodes (a) and (b) coauthored a publication in year 2003, nodes (c) and (a) coauthored a publication in year 2001 etc.

4.3 Tuning the Algorithms

In order to evaluate the results, the main variables of the algorithms and metrics proposed, have to be tuned so that the dataset information is better represented. The main targets of this tuning is reducing the memory and computing requirements without sacrificing any information. In addition, the variables of the algorithms are tuned to better represent the scores calculated. Variables such as temporal and structural decay, the number of hops in which neighbors are still considered similar to one another and even the time range in which algorithms are run are some of the many that have to be tuned to better extract the underlying knowledge of each dataset. In the next subsections, we analyse the best choices of values for all variables based on experimental results. The static version of SimRank was used to tune the variables that do not include time. Its evolving version uses it for each snapshot so everything that applies to the static version applies to the evolving version as well. Given that multiple experiments had to be executed and an experiment including the entirety of our datasets would take a long period of time, we calculated all changes to the datasets using a random sample consisting of 25% of the number of nodes in the original datasets. Choosing a subset of the nodes and not the edges, creates a representative set to tune the algorithms without removing essential structural information.

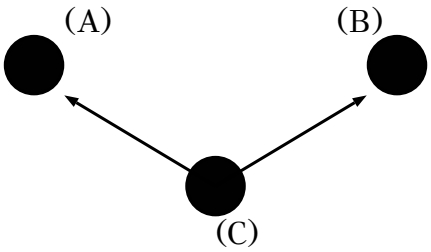
4.3.1 Neighbors

Due to the size and time requirement of a $n*n$ calculation on datasets of this size, the datasets have to be filtered and sampled without altering the underlying information. The first step is calculating how many hops an effective pair similarity calculation includes and how many neighbors included nodes should have.

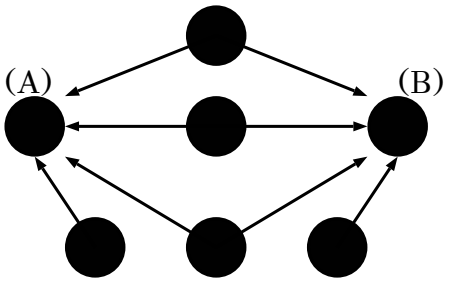
Degree threshold

It is essential to exclude nodes with degree lower than a threshold from our calculation. In many cases, node pairs are toping the similarity and metrics charts due to lack of other neighbors. For example, if two nodes only have a single, common neighbor, then their similarity will, almost certainly, be equal to the maximum allowed similarity. If we compare this to a pair of nodes that have four common neighbors pointing to both of them and one or more neighbors not pointing to them then, the results

are, as stated before, dominated and not representing the underlying information. Figures 4.3a and 4.3b better illustrate this example.



(a) Single Common Neighbor



(b) Multiple Common Neighbors

Figure 4.3: In figure (a), nodes (A) and (B) both have a single common neighbor, node (C), maximizing their similarity. In figure (b) the same nodes have multiple common neighbors and some uncommon ones, so their similarity will never reach the maximum value.

On all experiments conducted, nodes that represent structures like the one seen in Figure 4.3a completely dominate the structures that look like Figure 4.3b and thus a degree limit is introduced for all nodes in our datasets. Essentially, if a node has a degree of more than 6, it will not be included in the graph. Experiments showed that when 3 was the minimum degree, 100% of the top-k pairs had the minimum number of edges. At 4, this number dropped to 50% and at 6, it went down to 10%, which is considered sufficient due to it converging for every increment from that point on. As a result, a degree limit of 6 was introduced in our graphs. Figure 4.4 illustrates this.

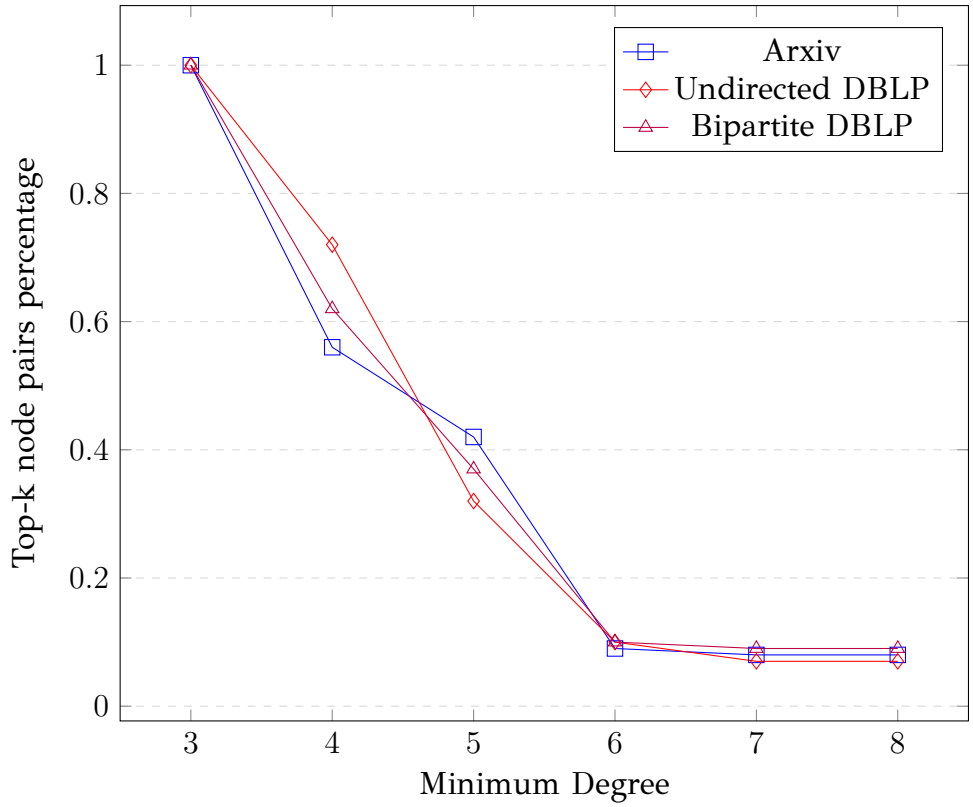


Figure 4.4: Percentage of top-k similarity pairs whose degree equals the minimum degree for all three datasets

Number of hops

In addition to node degree, the distance between two nodes can also be limited. Experimental results from the original paper [10] state this limit to be equal to 2. To make sure that the same number fits our data, we experimented with maximum hops equal to 3 and 4. Given that we have already excluded nodes with degree less than 6, going up to 4-hop neighbors can make the algorithm much more expensive to calculate. However, despite the complexity, in Figures 4.5, 4.6 and 4.7, it can be seen that the similarity of most 3-hop and 4-hop pairs is lower than the similarity of 2-hop pairs. Specifically, 90% of the 3-hop pairs have a lower similarity score than the 5th percentile of the 2-hop pairs. This percentage increases to 96% for 4-hop neighbors. Including more neighbors in our calculation might, in turn, change the similarity scores of all other nodes. This, however, didn't seem to be the case. Only two pairs of the 100 top pairs changed when including 3-hop neighbors to the calculation. Between including 3-hop and 4-hop neighbors, the top 100 list of pairs

didn't change at all.

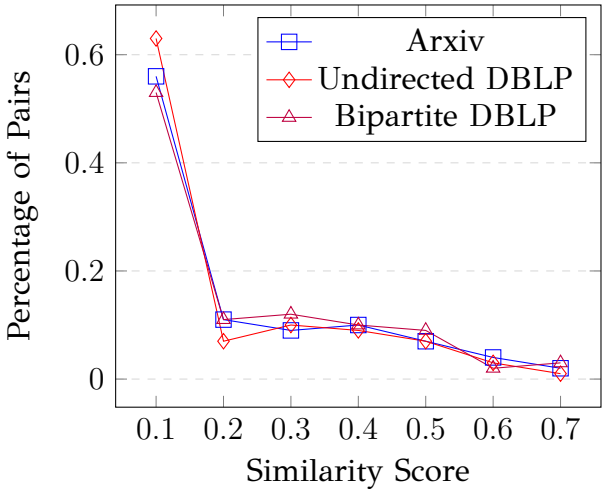


Figure 4.5: Pair percentage over similarity score using two-hop calculation.

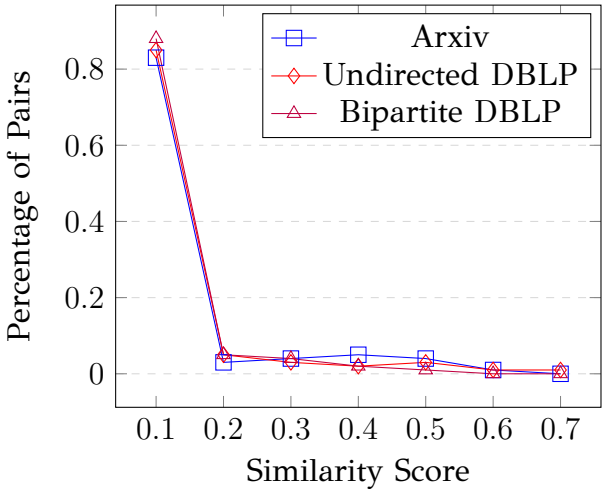


Figure 4.6: Pair percentage over similarity score using three-hop calculation.

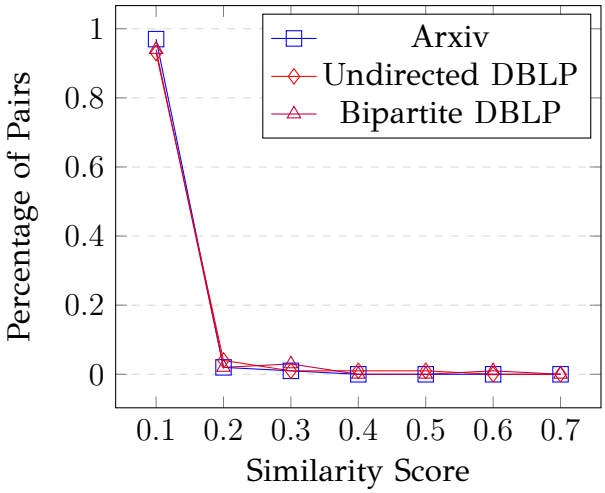


Figure 4.7: Pair percentage over similarity score using four-hop calculation.

4.3.2 Dataset Sample Size

Because the space and time complexity of many of the algorithms and metrics used is prohibiting the full analysis of large-size datasets (such as the one million edges of DBLP), sampling has to be used to limit the time required for the experiments to run and better filter the results. The removal of nodes that have a degree less than 6 led to a significant decrease in size for the Arxiv dataset. From 22.000 nodes and 350.000 edges, it went down to 9.000 nodes and 150.000 edges. An approximate 60% decrease. Given that a graph this size can be analyzed by most of the proposed algorithms in around 15 minutes (given the hardware used), no further dataset size reduction was conducted to not risk altering the results. In the case of undirected DBLP, however, the beginning 456.000 nodes and 1.4 million corresponding edges got reduced to 121.000 nodes and 700.000 edges by applying the 6-degree limit. In addition, the bipartite version of DBLP only had author nodes with less than 6 publications removed and had its numbers drop from 902.000 nodes and 1.3 million edges to 480.000 nodes and 610.000 edges. Both of the DBLP datasets required a sampling method due to their size. A reasonable variable to target for this, is time. It must be noted that only 9.000 authors (with more than 6 collaborations) exist in the DBLP dataset between 1959 and 2000. This means that 112.000 new people published their research in the past 16 years. This is equal to 7.000 new authors every year. To decrease them even further, a maximum of 5.000 publications were chosen at random from each year. This reduced the size of the dataset to 30.000

nodes and 110.000 edges which is manageable and representative of the data. For case studies conducted, no maximum number of publications was applied. The only limits applied were year limits and node degree ones. All results can be seen in Table 4.3.

Table 4.3: Final dataset node / edges count based on sampling conducted

	Arxiv	**DBLP**	
Type	*Directed*	*Undirected*	*Bipartite*
Original Size	22.000 / 350.000	456.000 / 1.400.000	902.000 / 1.300.000
low degree removed	9.000 / 150.000	121.000 / 700.000	480.000 / 610.000
5.000 per year chosen	- / -	30.000 / 110.000	130.000 / 230.000

4.3.3 Structural and Temporal Decay

The original SimRank paper (and all other relevant papers) suggests that a structural decay of $C = 0.8$ is used. Furthermore, a sufficient value must be found for the temporal decay introduced in the temporal versions of the suggested algorithm. 0.2, 0.4, 0.6, 0.8 and 1 where the main candidates for the temporal decay value. This choice depends on how much we require time to be essential to the calculation of the scores. We compared them by running the SimRank algorithm for all three datasets using all different values. The results can be seen per dataset in Figures 4.8, 4.9 and 4.10. The evolving version of SimRank was used to calculate the scores.

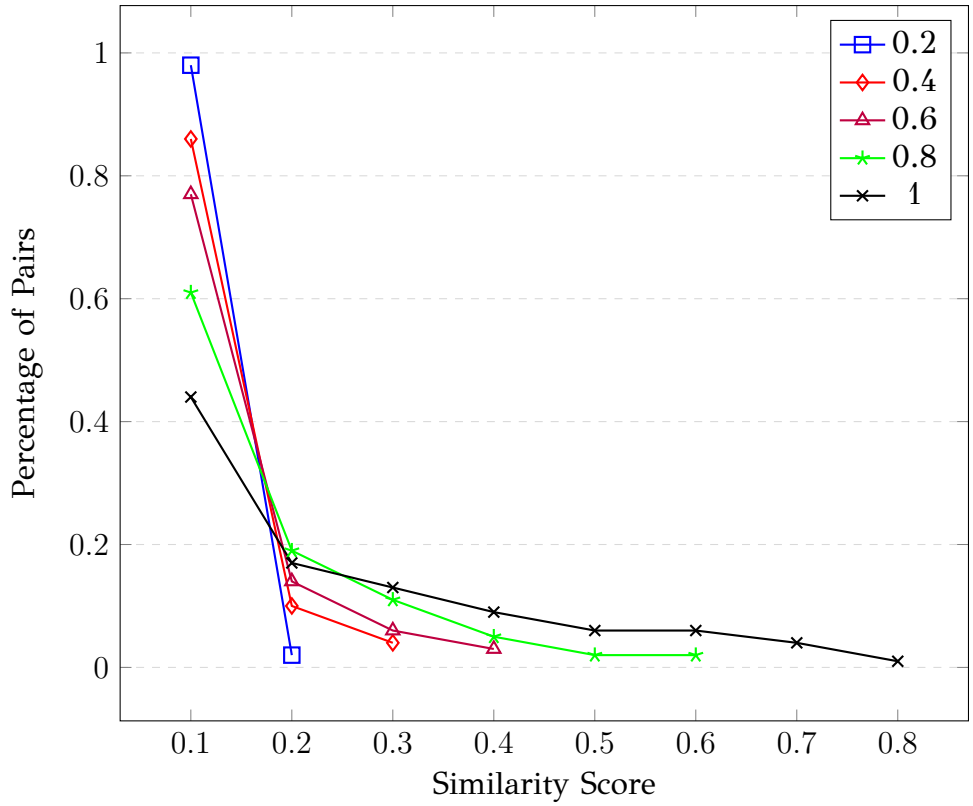


Figure 4.8: Similarity scores for each temporal decay value on the Undirected DBLP dataset.

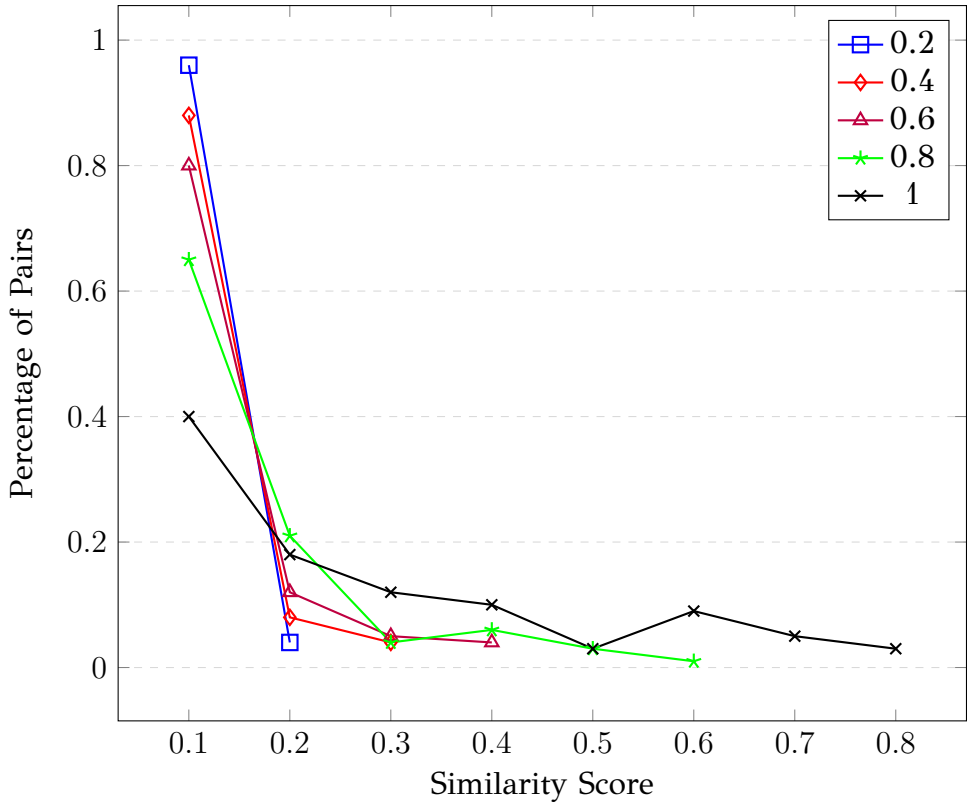


Figure 4.9: Similarity scores for each temporal decay value on the Bipartite DBLP dataset.

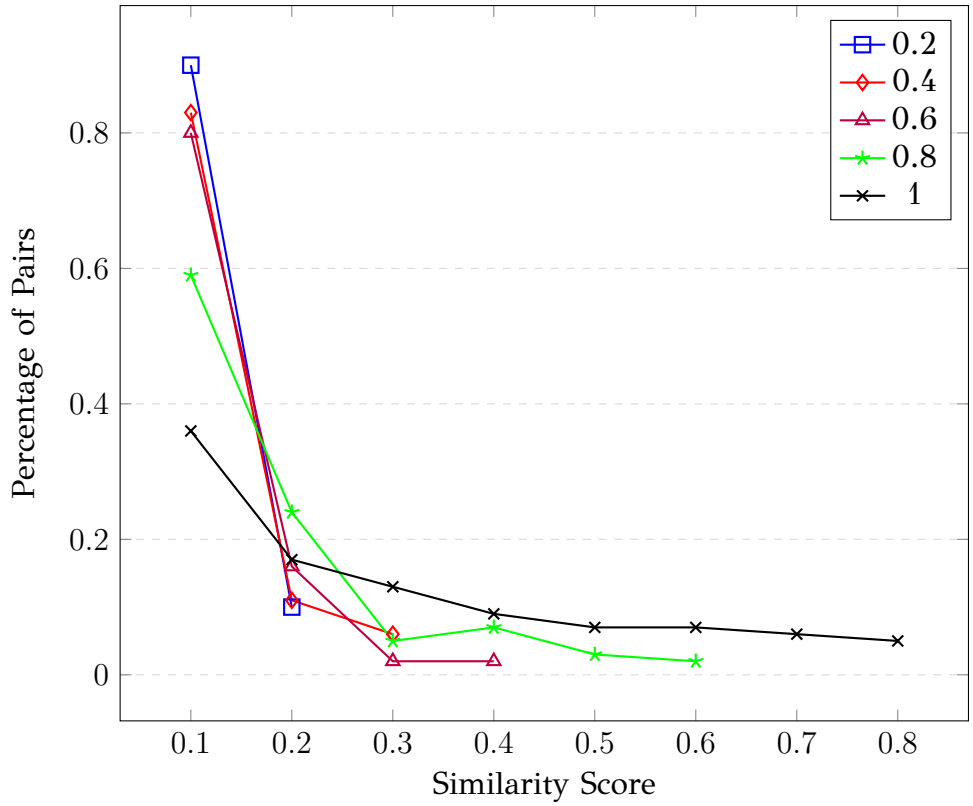


Figure 4.10: Similarity scores for each temporal decay value on the Directed Arxiv dataset.

It is apparent that having a greater temporal decay value increases the detail of the scores and forces a more equal distribution of similarity values. This does not, however, change the results. This means that the proportion of all pair scores remains the same (given that they are all multiplied by the same number, be it higher or lower). The value 1 cannot be used for our experiments because having a temporal decay of 1 means that time difference has no implication on scores. To have a better representation of all similarity values and being able to analyse the results in depth due to their distribution, the 0.8 value was chosen for the temporal decay variable.

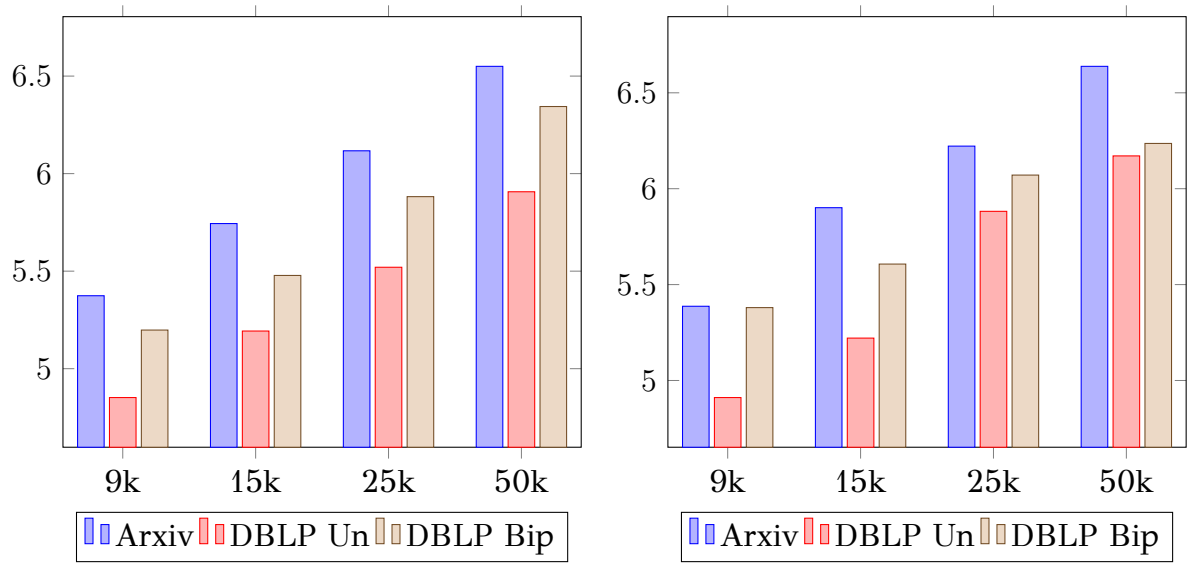
4.4 Static, Evolving and Temporal

4.4.1 Performance

Many of the algorithms used shared parts. To calculate the evolving SimRank we used the same algorithm core as its static counterpart and only used different algorithms to obtain node neighbors based on the temporal information. The same stands for all the evolving versions of distance, centrality and diameter where everything is calculated for each snapshot using the base algorithms and then, aggregated and averaged over all years the calculation took place in. On the other hand, all temporal versions of the algorithms used a different implementation core and so, performance changed drastically.

Graph creation

Because the same algorithms for creating the graph were used for all implementations, the only difference lies in the dataset structure and not in the algorithm analysing the dataset. However, some extra steps and loops were taken by some of the construction algorithms and thus, Figure 4.11a presents the construction times, for each dataset reading algorithm, for three different sizes. Figure 4.11b presents the same results but without setting the degree limit for each node. As can be seen, the inclusion of the degree limit does not affect the times. This is due to the fact that the graph is always constructed in its entirety and then re-parsed node by node and nodes that do not meet the requirements are removed. Despite node count, the inclusion or not of degree limits is irrelevant to the pre-processing time. Nonetheless, static, evolving and temporal algorithms, all used the same graphs created this way so further comparison bears no importance.



(a) Degree limit 6

(b) Degree limit 0

Figure 4.11: Construction times of all graphs for different node counts.

The whole graph is stored and all calculations are conducted exclusively in memory. This means that a low memory footprint is essential. The graph itself does not take up much memory. It is created using adjacency dictionaries that store each nodes neighbors and attribute dictionaries that store each nodes and edges attributes (such as time). The advantage of using dictionaries is that only the connections that exist can be created and no $n * n$ list is required. Figure 4.12 shows the base requirements, before running the algorithms, for the dataset sizes each implementation used. As stated before, Arxiv used 9.000 nodes and 150.000 edges. DBLP included 23.000 nodes and 100.000 edges. Memory-wise the undirected version of DBLP is the most effective, explained by the fact that it only stores the minimum number of edges. This happens because each author gets connected to few authors over the course of the calculation (only a few have reached 120 collaborations, the average node degree is way lower). On the other hand, in the Bipartite version, every author is connected to multiple papers (sometimes up to 220) and thus, the memory required for storing all those edges is way higher. Arxiv also has an increased average node degree due to most papers citing more than 20 other papers and it is apparent that the memory requirement is also higher than the undirected DBLP graph (the node and edge count is significantly lower than that of DBLP). For comparison, the figure also includes the memory required if no limitations were set on the dataset.

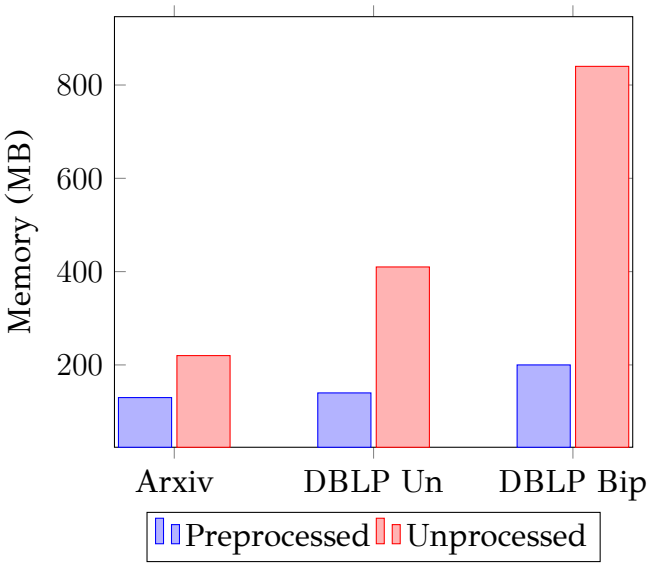


Figure 4.12: Memory requirements for each implementation.

Algorithm Execution

The SimRank algorithm is a slow algorithm that, even with all changes conducted, requires a significant amount of memory. Its static version is the most slow and heavy version on all three datasets due to the size of the dataset and available nodes / edges. Calculating the Evolving version of the algorithm drops time and memory requirements due to the size of snapshots and splitting / reduce of complexity of the calculation. The temporal calculation drops requirements even further. Figure 4.13 presents the results.

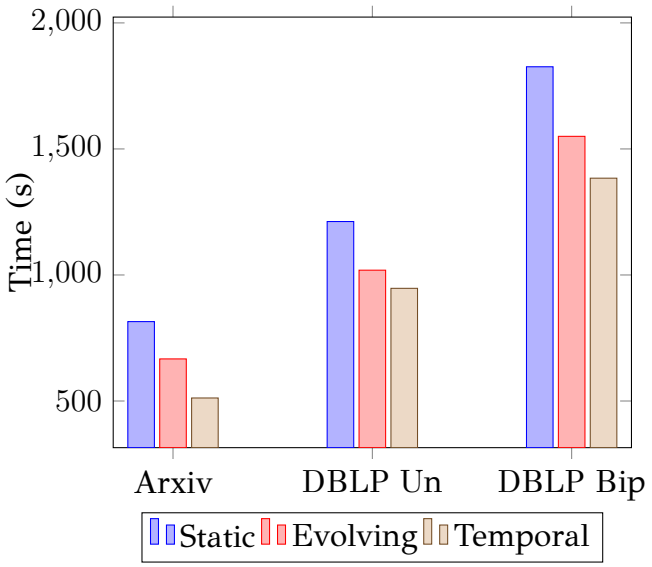


Figure 4.13: Execution times for all Simrank implementations

In the case of the other graph measures we analysed in the previous chapter, Distance and Degree Centrality are the only calculatable metrics in terms of time. Figure 4.14 presents the results for Distance calculations. All centrality calculations and the diameter calculation are all dependent on the calculation of distance, be it temporal or evolving. However, due to the high time and space complexity of all the algorithms (apart from degree centrality), their calculations are ommited.

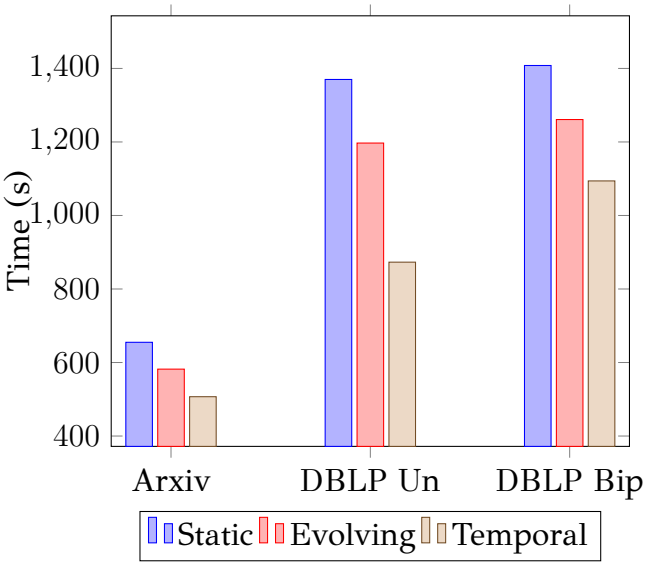


Figure 4.14: Execution times for all Distance implementations

Of course, memory is a huge issue when calculating any of the metrics used in this work. The structures used to store the results take up most of the space required by the algorithms and even after applying all limitations and optimizations require lots of MB to work. As is expected, the Static implementation requires the most memory to calculate due to the fact that there are more node pairs available whose distance has to be calculated. Evolving has less and Temporal even less than that. The high difference between undirected DBLP and bipartite DBLP is due to the bipartite implementations inclusion of publications in addition to the undirecteds authors. Results are presented in figure 4.15.

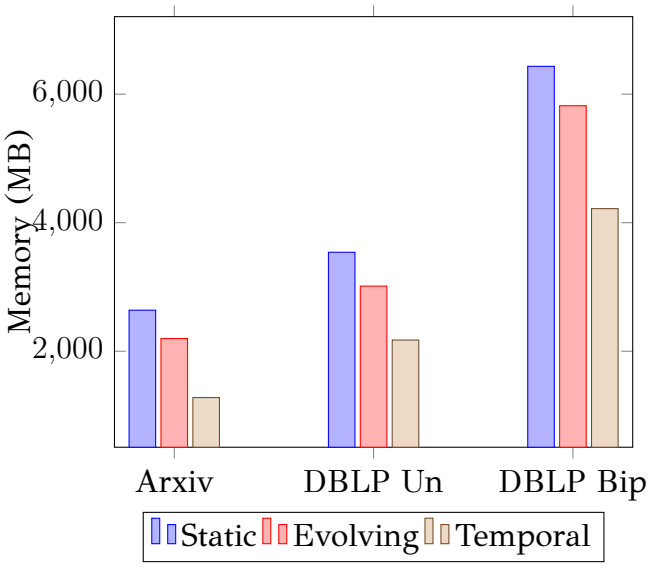


Figure 4.15: Memory requirements for all Simrank implementations

Similarly, 4.16 presents the memory requirements to calculate Distance in our graphs.

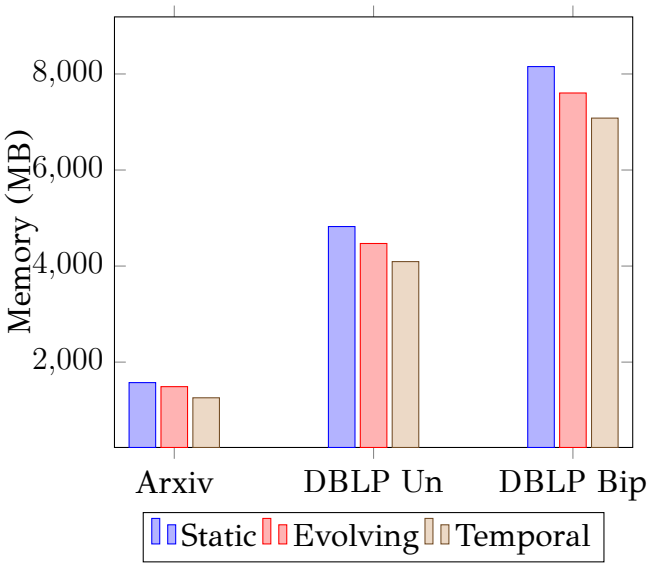
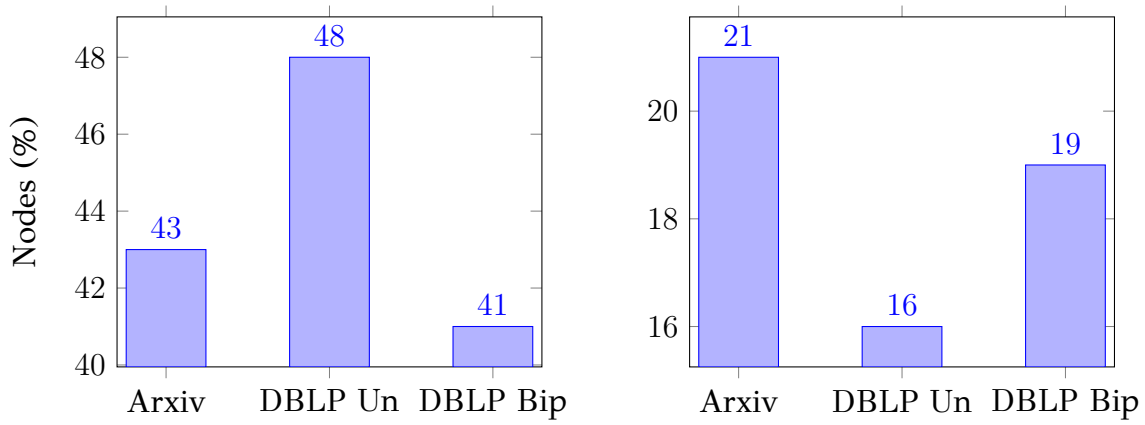


Figure 4.16: Memory requirements for all Distance implementations

4.4.2 Values

Memory and execution times are not the only changes of the algorithms. The main difference lies in the values of the node pairs. The significant difference is that 100% of the node pairs analysed by Static Simrank, Evolving SimRank and Temporal SimRank have different scores. This is expected because the calculation changes between all

three algorithms. However, the value changes of node pairs is significant. Between the Static and Evolving implementations of the algorithms 4.17a presents the percentages of nodes that moved from the top 50% of scores to the bottom 50%. Same percentages are presented for Evolving and Temporal implementations in figure 4.17b.

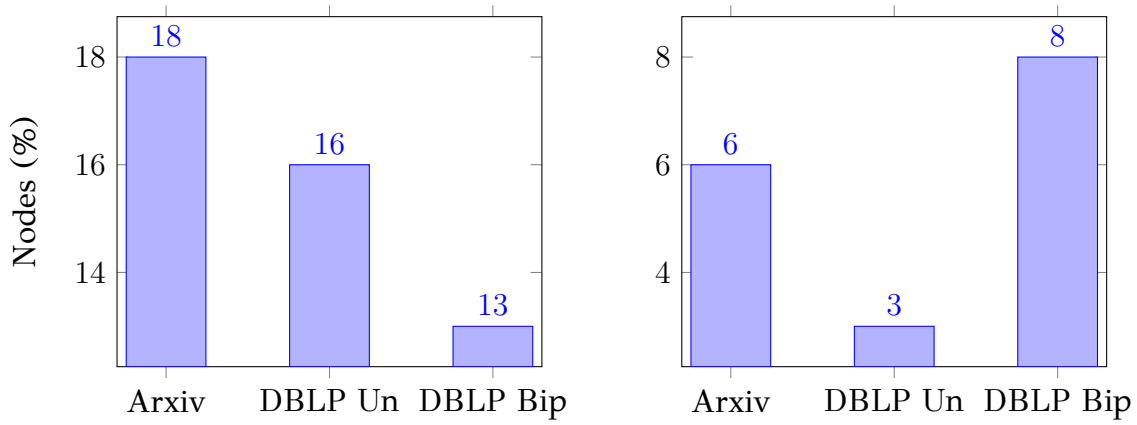


(a) Static to Evolving

(b) Evolving to Temporal

Figure 4.17: Percentage of nodes that moved between the top and bottom 50% of similarity scores

This strengthens the requirement to further analyze the temporal and evolving implementations and try to justify the changes. This becomes more apparent if we compare the same percentages for nodes that moved from the top 10% to the bottom 10% which is a huge drop or increase of score. Figure 4.18 presents the results. Finally, approximately 42% of the graph nodes are disconnected when calculating the distance in the evolving implementation using the snapshots and 69% of the graph nodes are disconnected when only accepting time respecting journeys. Also, more than 80% of the shortest paths between pairs that aren't disconnected are increased in size when using snapshots and more than 85% of them are increased using the temporal implementation. This is expected due to the fact that any non temporal shortest path is always a lower bound for any temporal or evolving distance.

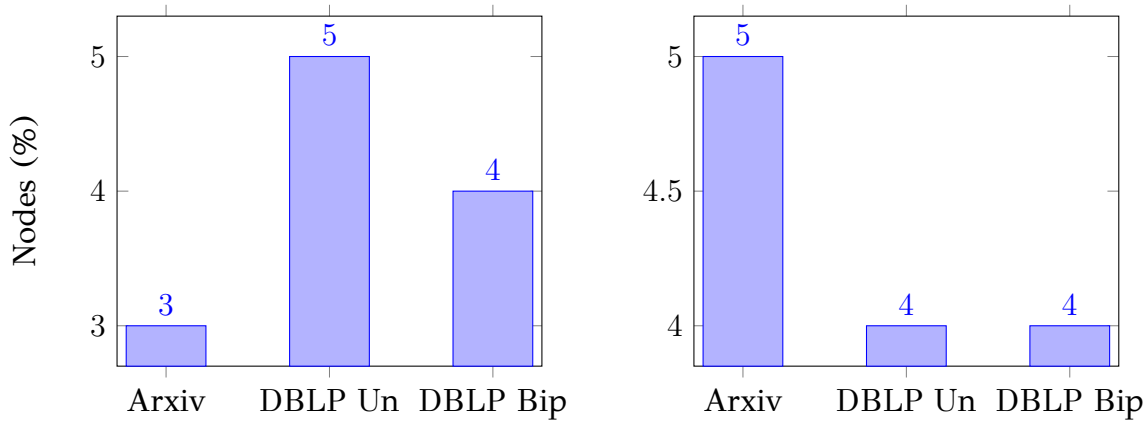


(a) Static to Evolving

(b) Evolving to Temporal

Figure 4.18: Percentage of nodes that moved between the top and bottom 10% of similarity scores

Finally, figures 4.19a and 4.19b show the percentage of nodes that had a change for less than 10% of their value moving between static, evolving and temporal implementations. It is apparent that moving away from a static calculation, towards an evolving one, completely changes the importance of node connections and enhances the temporal elements of the graph. This also shows how important temporal information can be to determining any measure on social graphs. SimRank is one of the many algorithms that can produce totally different results if the temporal dimension is accounted for.



(a) Static to Evolving

(b) Evolving to Temporal

Figure 4.19: Percentage of nodes that had their value change for less than 10%

4.5 Ranking the results

Ranking the resulting similarity scores and metrics, produced by the algorithms proposed and analyzed in the previous chapters, can provide us with important information about the datasets used, as well as the validity of the algorithms. Ranking, averaging, drops / increases and average values are only a few of the rankings that can be applied to the data. Below are the most representative results of the data used.

4.5.1 Top-k most similar

In the case of ranking the similarity of nodes using the static algorithm of SimRank the results are previewed in Tables 4.4 and 4.5.

Table 4.4: Top 10 Arxiv most similar publications using static SimRank

”Supersymmetry of Black Strings in D=5 Supergravities” & ”Finite Action in d5 Gauged Supergravity and Dilatonic Conformal Anomaly”
”On the Hyperbolic Structure of Moduli Spaces With 16 SUSYs” & ”String Universality”
”Mind the Gap” & ”Exact Renormalization Group Equations. An Introductory Review”
”On Open/Closed String Duality” & ”Constistency COnditions for Holographic Duality”
”Gauge Consistent Wilson Renormalization Group II: Non-Abelian Case” & ”Optimised Renormalisation Group Flows”
”Finite gravitational action for higher derivative and stringy gravities” & ”More on counterterms in the gravitational action and anomalies”
”Convergence of derivative expansions of the renormalization group” & ”Optimised Renormalisation Group Flows”
”Towards a loop representtation for quantum canonical supergravity” & ”M theory as a matrix extension of Chern-Simons theory”
”Conformal dynamics of quantum gravity with torsion” & ”Conformal anomaly of (2,0) tensor multiplet in d6 and AdS/CFT correspondence”
”PhreMology–calibrating M-branes” & ”Calibrated Geometries and Non Perturbative Superpotentials in M-Theory”

Table 4.5: Top 10 undirected DBLP author pairs using static SimRank

”Julio Ortega” & ”Alberto Prieto”
”David Grace” & ”Alexandros Kaloxylos”
”Nicolas Marin” & ”Maria J. Martin Bautista”
”Lalit M. Patnaik” & ”K. R. Venugopal”
”Karin Coninx” & ”Kris Luyten”
”Ester Bernado-Mansilla” & ”David E. Goldberg”
”Alexander Koller” & ”Massimo Poesio”
”Wendy Doube” & ”Thomas B. Hilburn”
”Veda C. Storey” & ”Jeffrey Parsons”
”Danny Dig” & ”Sarfraz Khurshid”

It should be noted here that most of the resulting pairs are in some cases authors

with some collaborations between them or that exist in the same field regardless of years of co-existence (some pairs have publications on the same field five or more years apart). This is natural due to the fact that we are not yet including time in the calculation and thus, just being connected with similar authors will increase the similarity.
Moving from the static to the evolving version changes the results. In the case of Arxiv, all of the publications, as in the static version, share the same theme but all top 10 pairs include publications that were published in a two year maximum range. In the static version, the year ranges were far greater (the pair in ninth place shared the same topic but had a 7 year difference). In the case of bipartite and undirected DBLP, the evolving version seems to work more efficiently. Most author publications line up much better temporally the closer we move to the final snapshot of the graph. The bipartite DBLP results can be seen in 4.6.

Table 4.6: Top 10 bipartite DBLP author pairs using Evolving SimRank

”Nachum Dershowitz” & ”Lior Wolf”
”Xabiel G. Paneda” & ”David Melendi”
”Matthew Hennessy” & ”Sophia Drossopoulou”
”Thilo Kielmann” & ”Aart van Halteren”
”Shai Shalev-Shwartz” & ”Ihab F. Ilyas”
”Marie-Laure Boucheret” & ”Ridha Bouallegue”
”Stathes Hadjiefthymiades” & ”Giannis F. Marias”
”Victor Maojo” & ”Jose Crespo”
”Ayoub Al-Hamadi” & ”Philipp Werner”
”Branislav Kusy” & ”Raja Jurdak”

It can be seen here that most of the pairs share many recent common co-coauthorships and collaborations. This shows that our algorithm is, indeed, counting the similarity correctly. Note though, that many share more than one common coauthors (research teams) that further enhance the score due to them being pointed by many nodes on many occasions.
In the case of the temporal version of SimRank, pair cases with long-term collaborations were really high in the score table. Results from the bipartite DBLP dataset using the temporal version of SimRank are listed in Table 4.7.

Table 4.7: Top 10 bipartite DBLP author pairs using Temporal SimRank

”Gunther Palm” & ”Ayoub Al-Hamadi”
”Sung-Gi Min” & ”Sachin Sharma”
”Christian Colombo” & ”Gerardo Schneider”
”Daniel Roviras” & ”Ridha Bouallegue”
”Maarten van Steen” & ”Spyros Voulgaris”
”Bin Li” & ”Jiwu Huang”
”Vishal M. Patel” & ”Rama Chellappa”
”George Bebis” & ”Mircea Nicolescu”
”Christoph Koch” & ”Immanuel Trummer”
”Branislav Kusy” & ”Raja Jurdak”

Here, there are fewer collaborations and more co-coauthorships. Most authors share common coauthors on different publications. However, this share is consistent through a long year range and thus, the temporal version of SimRank ranks them higher. The ones in the higher places share more recent co-coauthorships and thus have their scores further increased by that.

4.5.2 Drop / increase over time range

In the case of drop or increase the sets are dominated by pairs of authors that had many common co-authors over a time range and then stopped having any at all or vice versa. This was due to the author stopping the publication process altogether and not moving on to another field. Because of this, listing the author pairs doesn't present any important information due to this. Furthermore, this analysis cannot be conducted in publications from Arxiv due to their temporal uniqueness (a paper is published once and referenced in random intervals from then on).

4.5.3 Average Node Scores

The top 10 authors with the highest average node similarity with their one and two-hop neighbors are listed in Table 4.8. Additionally, in the case of venues, in the bipartite version of DBLP, there is some important information ranking them by the average similarity scores. Given that venues do not exist in the structure, for every

year in the time range, all publication similarity scores for that year were summed and the average score for each venue for each year was calculated. 10 venues were used for this: VLDB, ICDE, SIGMOD, EDBT, KDD, WWW, SIGIR, ICDM, CIKM, SDM. The calculation was conducted over 10 years (1995-2005). The resulting scores are analyzed in Figure 4.20.

Table 4.8: Top 10 nodes with the highest average node similarity using temporal SimRank on bipartite DBLP

”J. K. Aggarwal”
”Vijay Kumar”
”Dong Wang”
”H.Vincent Poor”
”Mubarak Shah”
”Ajay Kumar”
”Ying Wang”
”Barry L. Nelson”
”Tarek F. Abdelzaher”
”Larry S. Davis”

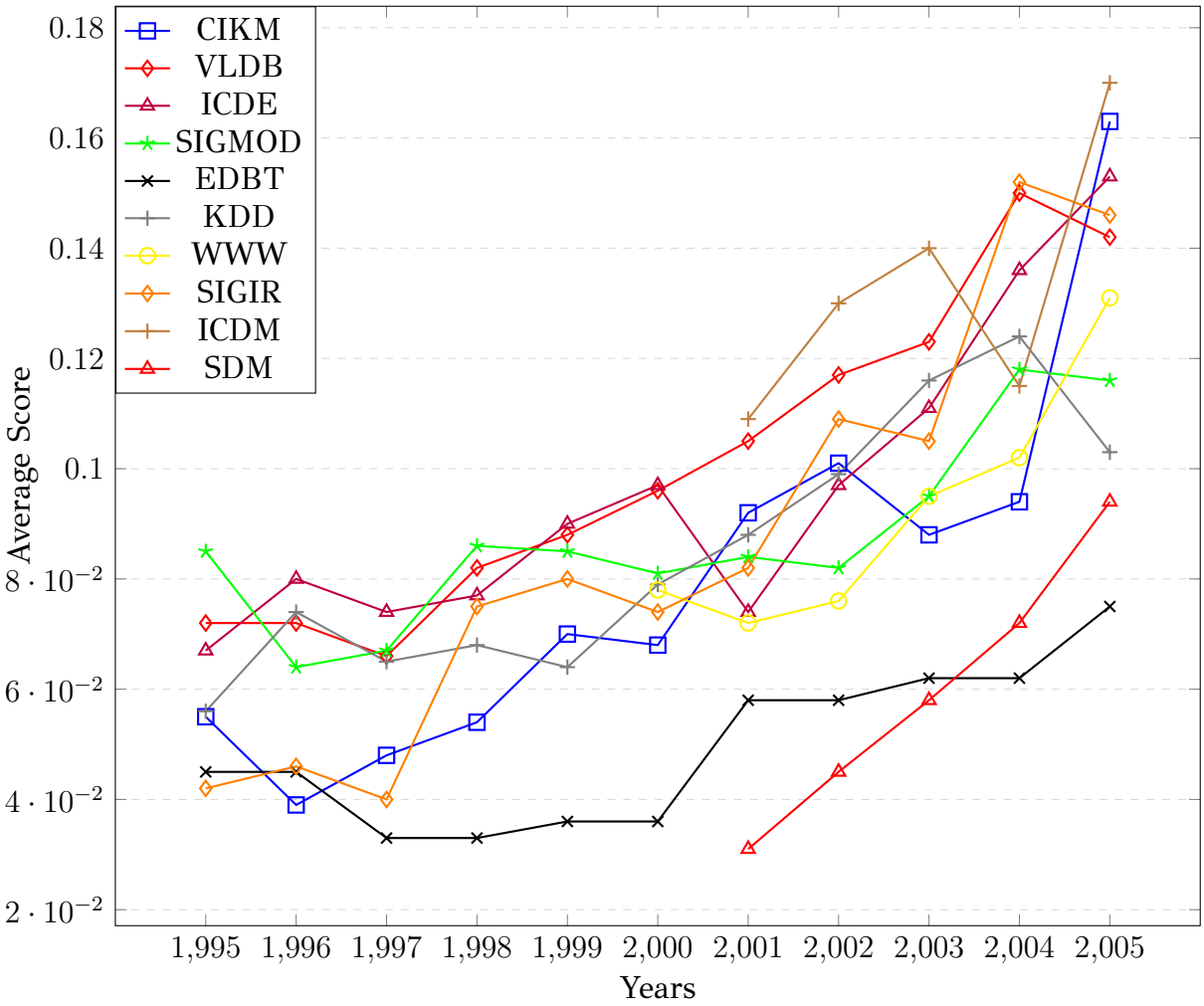


Figure 4.20: Venue similarity scores over the years 1995-2005 based on temporal SimRank on Bipartite DBLP.

4.5.4 Most similar neighbors count

To calculate the most similar neighbors count we need to set a threshold for the nodes that will be included in the ranking. We set this threshold equal to 0.2 given that there is, in all experiments, a sufficient portion of the similarity scores in that range (about 40%). Given that, Tables 4.9 and 4.10 present the resulting rankings.

Table 4.9: Nodes with neighbours whose similarity score exceeds 0.2 using Directed Arxiv and Temporal SimRank

Name	# neighbours with similarity >threshold
The Large N Limit of Superconformal Field Theories and Supergravity	310
Anti De Sitter Space and Holography	219
Gauge Theory Correlators from Non-Critical String Theory	208
Large N Field Theories, String Theory and Gravity	190
Monopole Condensation, And Confinement In N=2 Supersymmetric Yang-Mills Theory	172
Dirichlet-Branes and Ramond-Ramond Charges	165
String Theory Dynamics in Various Dimensions	147
M Theory As A Matrix Model: A Conjecture	142
Monopoles, Duality and Chiral Symmetry Breaking in N=2 Supersymmetric QCD	131

Table 4.10: Nodes with neighbours whose similarity score exceeds 0.2 using Bipartite DBLP and Temporal SimRank.

Name	# neighbours with similarity >threshold
Jake K. Aggarwal	26
Dong Wang	26
Alex Galis	25
Barry L. Nelson	25
Vipin Kumar	25
Ying Wang	25
Larry S. Davis	24
Qiang Ji	24
Mubarak Shah	23
Santosh Kumar	23

In the case of Arxiv, the publications found on the top ranks are both central papers to their respective fields and are cited by a really high number of publications. The corresponding number of nodes using the static SimRank is close to 200% higher which shows how much the similarity diminishes as the time between publication and citation increases. In the case of the Bipartite DBLP dataset a similar trend can be seen. All authors have a high amount of collaborations but the more recent the publications, the higher the score.

Chapter 5

Conclusion and future work

5.1 Contribution

This thesis introduces the problem of changing different algorithms, used to measure various metrics, such as pair similarity, graph diameter, pair distance and others to successfully be applied to graphs structured to incorporate temporal information. Solving this problem allows us to discover deeper relations between nodes and information concerning the underlying structure, that would otherwise be impossible to measure. Additionally, a methodology can be extracted that can be applied to various other algorithms so that similar results are achieved.

Along the way, a plethora of definitions, algorithms and implementations have been created, tuned and analyzed exploring the problem of changing a static algorithm to a temporal one in depth. Specifically, a formal definition has been given for both evolving and temporal versions of a pre-existing algorithm that measures similarity (SimRank), the definition of path has been changed to be applied in temporal information as a 'journey' and its inclusion in pre-existing measuring algorithms for centrality, diameter and distance has been analyzed. Furthermore, the suggested theory and changes were implemented using Python and the NetworkX library for

graph management and results were analyzed after applying the algorithms on three datasets of different structure and data and executing them entirely in the main machine memory.
Finally, representative results of each implementation were presented and ranked in various ways utilizing average values, value drops and increases over time ranges, finding the count of the most similar among the graph and others. All algorithms could not be tuned to fit all data so some generalizations were required but, given sufficient time, the modifications proposed can be tuned further and better.
Naturally, some of the proposed algorithms could not be executed on our datasets due to time and space complexity. Also, some of the proposed variables of the algorithms are considered to equal their base values in order to fit the dataset (a sufficient networks that includes traversal times and has sufficient structural and temporal information to accomodate it could not be obtained so traversal times are in all cases considered to be equal to one time instance).

5.2 Applications

This project has resulted in a documented, tested and fully working algorithm for measuring similarity on evolving and temporal networks. This algorithm can be used in various projects. It can enhance link prediction and suggestion engines, help with the statistical analysis of large graphs, introduce new ways of indexing information or be extended to work on more cases and more in depth. The measuring algorithms proposed can be tested given sufficient tuning and space / time resources. By tuning them, they can also be used in various research topics and extended to be incorporated in actual systems that require measuring of temporal information of networks.

5.3 Future Work

Apart from the already potential improvements / tunings and tests to the algorithms proposed, multiple future extensions can be proposed.
A first important extension would be including more algorithms and modifying them from static only implementations to temporal ones. Another, would be changing the

temporal implementation to be able to run without requiring recursive algorithms, while being able to maintain the whole graph history. This is rather difficult due to the potential space and time requirements of such a change. Additionally, the definition of 'journey' can be extended in multiple other algorithms that use paths and those algorithms can also be applied to temporal data. Finally, indexes based on scores can be created for faster look-up of the more 'central' or 'important' nodes of the networks (based on each definition).

Bibliography

[1] V. Kostakos, "Temporal graphs," *Physica A: Statistical Mechanics and its Applications*, vol. 388, no. 6, pp. 1007–1023, 2009.

[2] U. Khurana, "An introduction to temporal graph data management," tech. rep., Technical report, May, 2012.

[3] A. Campos, J. Mozzino, and A. Vaisman, "Towards temporal graph databases," *arXiv preprint arXiv:1604.08568*, 2016.

[4] V. Nicosia, J. Tang, C. Mascolo, M. Musolesi, G. Russo, and V. Latora, "Graph metrics for temporal networks," in *Temporal Networks*, pp. 15–40, Springer, 2013.

[5] N. Santoro, W. Quattrociocchi, P. Flocchini, A. Casteigts, and F. Amblard, "Time-varying graphs and social network analysis: Temporal indicators and metrics," *arXiv preprint arXiv:1102.0629*, 2011.

[6] R. Angles and C. Gutierrez, "Survey of graph database models," *ACM Computing Surveys (CSUR)*, vol. 40, no. 1, p. 1, 2008.

[7] S. Jouili and V. Vansteenberghe, "An empirical comparison of graph databases," in *Social Computing (SocialCom), 2013 International Conference on*, pp. 708–715, IEEE, 2013.

[8] N. Agarwal and D. Mahata, "Grouping the similar among the disconnected bloggers," *Social Media Mining and Social Network Analysis: Emerging Research: Emerging Research*, p. 54, 2013.

[9] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[10] G. Jeh and J. Widom, "Simrank: a measure of structural-context similarity," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 538–543, ACM, 2002.

[11] S. Rothe and H. Schütze, "Cosimrank: A flexible & efficient graph-theoretic similarity measure.," in *ACL (1)*, pp. 1392–1402, 2014.

[12] D. Fogaras and B. Rácz, "Scaling link-based similarity search," in *Proceedings of the 14th international conference on World Wide Web*, pp. 641–650, ACM, 2005.

[13] Y. Sun and J. Han, "Ranking methods for networks," in *Encyclopedia of Social Network Analysis and Mining*, pp. 1488–1497, Springer, 2014.

[14] G. Kossinets and D. J. Watts, "Empirical analysis of an evolving social network," *science*, vol. 311, no. 5757, pp. 88–90, 2006.

[15] P. Rozenshtein and A. Gionis, "Temporal pagerank," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 674–689, Springer, 2016.

[16] M. S. Mariani, M. Medo, and Y.-C. Zhang, "Ranking nodes in growing networks: When pagerank fails," *Scientific reports*, vol. 5, 2015.

[17] B. Bahmani, R. Kumar, M. Mahdian, and E. Upfal, "Pagerank on an evolving graph," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 24–32, ACM, 2012.

[18] G. Kossinets and D. J. Watts, "Origins of homophily in an evolving social network," *American journal of sociology*, vol. 115, no. 2, pp. 405–450, 2009.

[19] H. Tong, S. Papadimitriou, S. Y. Philip, and C. Faloutsos, "Proximity tracking on time-evolving bipartite graphs,"

[20] "Centrality - wikipedia." `https://en.wikipedia.org/wiki/Centrality`. (Accessed on 02/15/2017).

[21] "Kdd cup 2003 - datasets." `https://www.cs.cornell.edu/projects/kddcup/datasets.html`, August 2003. (Accessed on 01/11/2017).

[22] "dblp: computer science bibliography." `http://dblp.uni-trier.de/`. (Accessed on 01/11/2017).

Author's Publications

[1] Kermanidis, K. L., Karydis, I., Koursoumis, A., & Talvis, K. (2014). Combining Language Modeling and LSA on Greek Song "Words" for Mood Classification. International Journal on Artificial Intelligence Tools, 23(02), 1440007.

[2] Brilis, S., Gkatzou, E., Koursoumis, A., Talvis, K., Kermanidis, K. L., & Karydis, I. (2012, September). Mood classification using lyrics and audio: A case-study in greek music. In IFIP International Conference on Artificial Intelligence Applications and Innovations (pp. 421-430). Springer Berlin Heidelberg.

[3] Koursoumis, A., E., Founta, A. M., Talvis, K., Mprilis, S., ... & Kermanidis, K. L. (2012, May). Learning to case-tag modern greek text. In Hellenic Conference on Artificial Intelligence (pp. 353-360). Springer Berlin Heidelberg.

Short Biography

I grew up loving computers and everything related to modern technology.

I graduated from the Informatics Department of the Ionian University, in Greece, in February 2014. On the same month, I joined the computer science departments post-graduate program in the University of Ioannina and am working on getting my Master's degree since.

I worked one year for Senseworks Ltd. as a full stack web developer and since then I work as a freelance web developer. I have been a part of both research groups and software development groups and grew to like the latter more. However, my experience on the former helped me evolve in many ways.

I generally find myself working better and more efficiently in a team rather than alone.

On my spare time I like going out with friends, watching movies and tv-series, supporting the local football team and playing games, be it video games or tabletop pen and paper ones.

I am also a member of the Pan-hellenic Philanthropic Association "Bread & Action" and help collect and distribute donated goods to families living in poverty in Epirus, Greece.