MASTER'S THESIS 2023

Finding Candidate Node Pairs for Link Prediction at Scale

Filip Kalkan, Mahir Hambiralovic

ISSN 1650-2884
LU-CS-EX: 2023-04

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-04

Finding Candidate Node Pairs for Link Prediction at Scale

Att Hitta Nodpar för Länkprediktion i Enorma Grafer

Filip Kalkan, Mahir Hambiralovic

Finding Candidate Node Pairs for Link Prediction at Scale

(Evaluating existing methods and proposing a novel method, based on Personalized PageRank, for performing Link Prediction efficiently on large graphs)

Filip Kalkan
fi1231ka-s@student.lu.se

Mahir Hambiralovic
mahir.hambiralovic@gmail.com

January 20, 2023

Master's thesis work carried out at Neo4j Inc..

Supervisors: Per Andersson, per.andersson@cs.lth.se
Adam Schill Collberg, adam.schill-collberg@neo4j.com

Examiner: Alma Orucevic-Alagic, alma.orucevic-alagic@cs.lth.se

Abstract

There are methods for inferring whether a pair of people are likely to become friends or whether two kinds of drugs are likely to interact if consumed simultaneously. The methods solve the problem of link prediction, i.e. answer the question "Is a link (friendship, interaction) likely to form between two particular nodes (people, drugs)?". Generalizing the problem to graphs translates it to predicting if particular node pairs are likely to form links. As predicting links between all possible node pairs is computationally infeasible for larger graphs, methods for narrowing down the search space are required to efficiently solve the problem.

We propose a novel algorithm, DAPPR, for resolving this issue and compare it against an existing solution LinkWaldo, along with breadth first search and a variant of KNN. The algorithms are evaluated by their ability of finding hidden edges on on real-world graphs, and it is shown that DAPPR outperforms all compared algorithms.

Keywords: MSc, Graph, Link Prediciton, Large Scale Link Prediction, Candidate Node Pairs, Candidate Node Pair Selection, LinkWaldo, Nearest Neighbor Descent, NN-Descent, Personalized PageRank, Breadth First Search, Graph Machine Learning, DAPPR, Distributed Approximate Personalized PageRank

Acknowledgements

We would like to thank Adam Schill Collberg, Florentin Dörre and Jacob Sznajdman at Neo4j for their guidance, support and feedback, and for going the extra mile to make this work the best possible.

Contents

Chapter 1

Introduction

By its simplest definition, a graph is any structure with objects where some pairs of objects have some relationship. Objects are often interchangeably referred to as nodes or vertices and relations as edges or links [4]. With this broad definition, it is easy to see how graphs are a useful representation for many complex systems, such as social networks, energy networks, routing systems, etc., and that it has many useful mathematical properties for finding e.g. shortest paths between nodes or identifying central nodes. Although graph theory has been essential in mathematics for a long time, graphs have been late to the party in the internet age [21]. With the emergence of efficient and practical graph database technologies such as Neo4j, the number of applications using graph representation natively has increased dramatically, making it the fastest growing type of database [19][6] and enabled new innovations in the field to emerge, such as applying Machine Learning on the graph context. One field that has been of particular interest is link prediction. Link prediction is the task of predicting new, unseen, edges in a given graph [18]. The problem can be formulated as either:

- **Completing an incomplete graph**. An example of this would be a graph of drug-to-drug interactions, where it is obviously incomplete as not all drug combinations have been tested. Here the aim would be to discover new drug-to-drug interactions that have not yet been discovered. A 2018 paper by Zitnik, Agrawal and Leskovec did just this and managed to find documented cases for 5 out of 10 drug-to-drug interactions made by the prediction model [28].

- **Finding future links in a temporal graph**. As some graphs can be represented as growing over time, models can be trained to predict new links at any given time step. One such application would be predicting future relationships in a social network [16].

1.1 Problem Statement

While much research has been devoted to finding methods that predict the most likely edges to form (or that should exist already) [18][16] [9] [17] [2], many of these approaches only run a prediction on the given node-pairs you ask it to predict. To predict all new edges in any given graph, one would have to run the method on every non-adjacent node pair possible in the graph, from here on referred to as exhaustive candidate selection. This task can become cumbersome on larger graphs, as the number of possible node pairs in a graph is upper bounded by n^2 if the edges are directed (where n is the number of nodes) and $\frac{n^2}{2}$ if they are undirected. Since many real-world graphs are generally sparsely connected, they have complexity $O(n^2)$. As performing n^2 operations on very large graphs is technically infeasible, an approach for selecting which subset of node pairs the prediction model should consider is necessary.

Mathematically, we formulate the candidate selection problem (similar to Caleb et.al. [3]) as: *Given a graph $G = (V, \mathcal{E})$ and a budget $c \ll n^2$, return a set of plausible candidate node pairs $\mathcal{P} \subset V \times V$ of size $|\mathcal{P}| \leq c$ for a link prediction model to make decisions about.*

1.1.1 Aim

The aim of this thesis is to evaluate existing and novel efficient algorithms that, given a graph, return a set of plausible candidate node pairs for a link prediction model to run upon.

1.1.2 Limitations

We limit the scope of this thesis to graphs which

- have only one node type,
- have only one edge type,
- edges are unweighted (a node pair can be linked by up to one edge),
- do not contain self loops,
- represent real world problems (as opposed to synthetic graphs).

1.2 Contribution

While the topic of link prediction has been extensively studied, many of the papers published on the topic only consider the time complexity of performing link prediction on node pairs composed of a specific node (query node) in conjunction with each of the remaining nodes in the graph [18]. Furthermore, they have time complexities of $\Omega(n)$ [18] as there are n nodes to consider linking to the query node, and thus running it on the entire graph (considering each possible query node) requires an additional multiplication of n, resulting in quadratic time complexity $O(n^2)$. Therefore, considering only one query node at a time falls short of the problem if one wishes to infer possible edges among all non-adjacent nodes in a large

graph efficiently. This thesis aims to investigate and suggest algorithms for efficiently running link prediction on large graphs, mainly by reducing the search space and performing link prediction on only a fraction of the possible node pairs.

Individual contributions to this thesis are listed in appendix C.

1.3 Original Source Code

The original source code of our findings and implementations can be found at `https://github.com/neo4j/dappr`.

Chapter 2
Background

This chapter sets the theoretical framework for graph notation, presents some established link prediction methods and other tools related to the topic of candidate node pair selection, concluding with a literature study of previous work on the topic.

2.1 Preliminaries

Let $G = (V, \mathcal{E})$ be a graph consisting of the nodes in the set V and the edges $(v, u) \in \mathcal{E} \subseteq V \times V$. The neighbors of a node v are denoted $N(v)$ and $u \in N(v)$ such that $(v, u) \in \mathcal{E}$. The number of nodes in the graph $|V|$ is referred to as n. The degree of a node v is defined as its number of neighbors $|N(v)|$, and d is defined as the average degree of all nodes in the graph ($\frac{|\mathcal{E}|}{n}$). Furthermore, the distance $dist(u, v)$ between two nodes u and v is considered to be the number of edges included in the shortest path between the two nodes.

A common way to represent a graph is through its *adjacency matrix*, which is a $|V| \times |V|$ matrix where each row and column represents a node index, and the elements are 1 (given that the edges are unweighted) if there is an edge between the two node indices and 0 otherwise.

The major mathematical symbols used throughout this thesis are summarized in appendix A.1.

2.2 Node Embedding

A node embedding x_i is a real valued vector representation of a node $v_i \in V$.

The goal of embedding nodes is to project the nodes onto points in a latent feature space where geometric relations reflect the interactions in the graph [12]. For instance, node embeddings can capture structural information about the nodes' neighborhoods (also called topological information) and can capture properties of the nodes or edges if those are available. Downstream, the embeddings are usable as rich inputs to machine learning tasks that

capture information that would otherwise be lost by only using simple properties such as a node's number of neighbors.

2.3 Link Prediction Models

This section describes a selection of common link prediction models composed of proximity models (PM) and machine learning models.

PMs are used to determine the similarity between two nodes in a graph. These are similarity based scoring functions that measure some form of distance and/or commonality between two nodes' neighborhoods. These models do not necessarily predict if a node pair should be connected by an edge or not, but rather output a score which can then be compared against other pairs' scores. Some of these have, however, been demonstrated to be fairly successful for link prediction [1][18][3]. Machine learning based models, presented lastly, output a prediction for whether a link between a pair of nodes should exist. These models are able to find less obvious implicit relations in comparison to the PMs, and can therefore be tailored to the unique attributes of each graph. An example of such relations would be finding similarities between sub-graphs to learn better link prediction-related properties on those, or finding the role that strongly connected nodes have in the graph and their role in the link prediction context.

2.3.1 Common Neighbors

One of the simplest PMs is Common Neighbors (CN), which returns a score based on the number of common neighbors between two nodes [18].

$$sim_{CN}(u, v) = |N(u) \cap N(v)| \tag{2.1}$$

2.3.2 Adamic-Adar

Proposed in a 2003 paper by Adamic and Adar [1], Adamic-Adar (AA) is an extension of CN in which the value of each common neighbor is scaled against the degree of the common neighbor. Though a simple function, AA has been proven to be successful in link prediction in multiple settings [1][18][3].

$$sim_{AA}(u, v) = \sum_{z \in N(u) \cap N(v)} \frac{1}{\log |N(z)|} \tag{2.2}$$

2.3.3 Jaccard Similarity

Jaccard Similarity (JS) intuitively extends on CN, taking into account the fraction of u and v's neighborhood which are common neighbors [18].

$$sim_{JS}(u, v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|} \tag{2.3}$$

2.3.4 Machine Learning Models

Link prediction can be defined as a machine learning problem. This definition has been explored with promising results using many different methods, ranging from ensembles of heuristic methods as the ones mentioned above [17] to more sophisticated models such as K-Nearest Neighbors, Support Vector Machines, Naive Bayes, Neural Networks and Graph Neural Networks [18][26]. Many of these approaches leverage topological information in the graph through the use of embeddings.

An instance of utilizing machine learning models for link prediction was presented by Lichtenwalter et. al [17], where an ensemble approach was used. Specifically, the approach made use of a random forest on features used in traditional link prediction methods and including some heuristic similarity scores in the ensemble, e.g. node degrees, shortest path, AA and JS scores. The authors find that it outperforms existing methods (such as AA, CN, Katz, PropFlow) by more than 30% in AUC. The authors conclude with a strong recommendation for using supervised methods for being superior in a range of datasets, without requiring any domain-specific knowledge on the given graph.

2.4 Personalized PageRank

First proposed by Page et. al. in 1999 as the major technology behind the search engine's ranking of search results, PageRank is an algorithm for finding the importance of different nodes in a graph [23]. Although the authors show that the problem can be defined in multiple ways (e.g. eigenvalue decomposition), they present an equivalent, yet more intuitive explanation of the algorithm as the probability of a random surfer. This approach explains PageRank as a random surfer browsing across the graph (the web in this case) and randomly moving across edges (hyperlinks), with some probability of teleporting to a random node at any time, while counting the number of visits on each node (website) and finally outputting a probability distribution of visiting any given node. This method is referred to as random walk with restarts, where the surfer has some probability $1-\alpha$ (where $\alpha \in [0, 1]$) of teleporting to any node in the graph during each step. The algorithm proved to be not only useful for searching the web (laying the foundation of the modern search engine Google) but has been successfully been applied to e.g. predicting the expression of genes in GeneRank, evaluating the importance of brain regions, finding root causes in distributed systems and predicting traffic flows [10].

Extending on an absolute ranking for the entire graph, the authors of PageRank also mention a special case of PageRank called Personalized PageRank (PPR), where the random surfer, in the event of a teleportation, always teleports to a given query node. This modification could be used for e.g. *"personal search engines"* [23]. Although the original PageRank implementation was based on eigenvalue computation, the Random Walk with Restarts, or random surfer, approach can be used to compute PageRank (or PPR) [27] using the following iterative definition

$$\pi_q(t+1) = \alpha P^T \pi_q(t) + (1-\alpha)e_q, \tag{2.4}$$

where $\pi_q(t)$ is a probability distribution over the nodes in the graph, representing the probability of a random walker's location in the graph at time step t with query node q. $\boldsymbol{P}$ (also called transition matrix) is a $|V| \times |V|$ matrix containing the transitional probabilities be-

tween all node pairs. The element at row x and column y is computed as $p_{xy} = \frac{a_{xy}}{|N(x)|}$ where a_{xy} is the corresponding element in the adjacency matrix. The vector e_q of dimensions $1 \times |V|$ represents the query node as the element corresponding to the query node has value 1 and all other elements have value 0 [27].

$\pi_q(t)$ converges to the PPR for q as $t \to \infty$. The stable state $\pi_q = \pi_q(t)|_{t\to\infty}$ can be expressed as

$$\pi_q = (1 - \alpha)(I - \alpha P^T)^{-1} e_q \tag{2.5}$$

where I is the identity matrix.

PageRank and PPR have previously been applied to the link prediction problem with some success[26][20], however, we have not found any published resources of their use in the link prediction candidate selection problem.

2.5 Related work

The following section contains a literature study that acts as the foundation for selecting the candidate selection methods to benchmark.

2.5.1 Nearest Neighbor Descent

In a 2011 paper by Dong et. al., a method called Nearest Neighbor Descent (NN-descent) was proposed for an efficient approximate implementation of finding the K-Nearest Neighbors (KNN) of all nodes in a graph, given any similarity measure [7]. While an exact implementation of KNN with *"brute-force has cost $O(n^2)$ and is only practical for small datasets"*, the purpose was to create an approximate KNN-method that was general, scalable, space efficient, fast and accurate, and easy to implement. The proposed method called NN-descent is based on the assumption that *"a neighbor of a neighbor is also likely to be a neighbor"*, and, excluding some optimizations and variations, can be summarized as:

1. Define the constant $k \in \mathbb{N}^+$ representing the number of sought nearest neighbors for each node, and a similarity function $sim : V \times V \to \mathbb{R}^+$.
2. Select k random neighbors for each node $v \in V$, creating $KNN[v]$.
3. For each $v \in V$ compute the set $KNN_{new}[v] = \{(w, sim(v, w)) \mid u \in KNN[v] \wedge w \in KNN[u]\}$.
4. For each $v \in V$, replace the set $KNN[v]$ with the K elements with highest score in $KNN[v] \cup KNN_{new}[v]$.
5. Repeat step 3-4 until the KNN-sets for all nodes have converged.

The proposed method leverages some techniques to improve efficiency, while proving to not sacrifice recall significantly. The primary improvements are local joins, sampling and early termination. Local joins reduce the number of evaluations in half by only evaluating (a, c) once in the subgraph $a-b-c$ when b is evaluated in the iterative step, rather than evaluating (a, c) on a's iterative step and then (c, a) on c's iterative step, as in the original implementation. Sampling allows for a further reduction in the number of evaluations by only comparing

a sampled subset of new, unevaluated nodes in $KNN[v] \mid v \in V$, using sample rate ρ, rather than the entire list each time. Finally, NN-descent uses early termination, allowing the user to set a convergence threshold δ to terminate the algorithm early. When evaluating NN-descent experimentally, using a fixed $k = 20$, the authors find empirically a time complexity of $O(n^{1.14})$ on a number of datasets, with a scan rate (number of comparisons / n^2) that is lower than 0.05 in most cases, and shrinks as the dataset grows.

In NN-descent a collection $KNN[v]$ of k nodes is kept for each node v. ρk nodes are sampled at each iteration. These sampled nodes are compared against all other nodes in $KNN[v]$, corresponding to $O(k)$, where k is set to $\lfloor \frac{c}{n} \rfloor$ operations in our experiments. As this is done for n nodes at each iteration, we conclude that the time complexity of NN-descent is $O(nk^2) = O(c^2/n)$.

Since the proposed algorithm can be used for any given similarity measure, it can be generalized to any problem that seeks to output a set of k node pairs for each node in a graph, using some function $sim : V \times V \rightarrow \mathbb{R}^+$. It is therefore possible to run NN-descent using some link prediction method as its similarity function. This yields an approximation of the k most likely new neighbors for each node. A solution based on the same KNN assumptions is applied to large scale link prediction by Neo4j in the Graph Data Science library [22].

2.5.2 Non-Negative Matrix Factorization with Bagging

One of the earliest attempts to solve the link prediction at scale problem was made by Duan et. al. in 2017 [9]. The authors proposed a method for splitting the dataset into smaller chunks using bagging methods in order to reduce the quadratic problem into $O((\frac{n}{b})^2)$, where b is the number of bags. They then borrow concepts from the problem of missing value estimation and collaborative filtering (e.g. recommender systems) to predict missing links within each bag, or subgraph. As collaborative filtering often relies on imputing missing values in a matrix, these concepts could be applied for finding missing values in an adjacency matrix (i.e. imputing missing edges). The authors settle on using Non-Negative Matrix Factorization, a type of Latent Factor Model [9] and select the k best pairs. In the experimental results, the authors conclude that the model scales *"nearly linearly with the increase of n"*. Finally, they compare the precision of the method against AA, Resource Allocation and Cluster Affiliation Model for large graphs (using the same bagging method as the proposed method). Non-Negative Matrix Factorization with bagging is concluded to have the best precision in 2 out of 3 datasets, where it is outperformed by AA in one dataset. Duan et. al. selected precision as the main evaluation metric, thus implying their target is to output a relatively small set of final predictions.

2.5.3 LinkWaldo

In 2022, Caleb et.al. proposed one of the first methods for solving the problem of efficiently finding a subset of all possible node pairs to perform link prediction on [3]. In the paper, they describe a method called *LinkWaldo* which leverages node groupings, bayesian statistics and Locality Sensitive Hashing (LSH), a family of fast approximate KNN algorithms [24]. In essence, LinkWaldo outputs a collection of node pairs such that a pair of nodes has similar

embeddings and belong to node classes that are strongly connected in the graph.

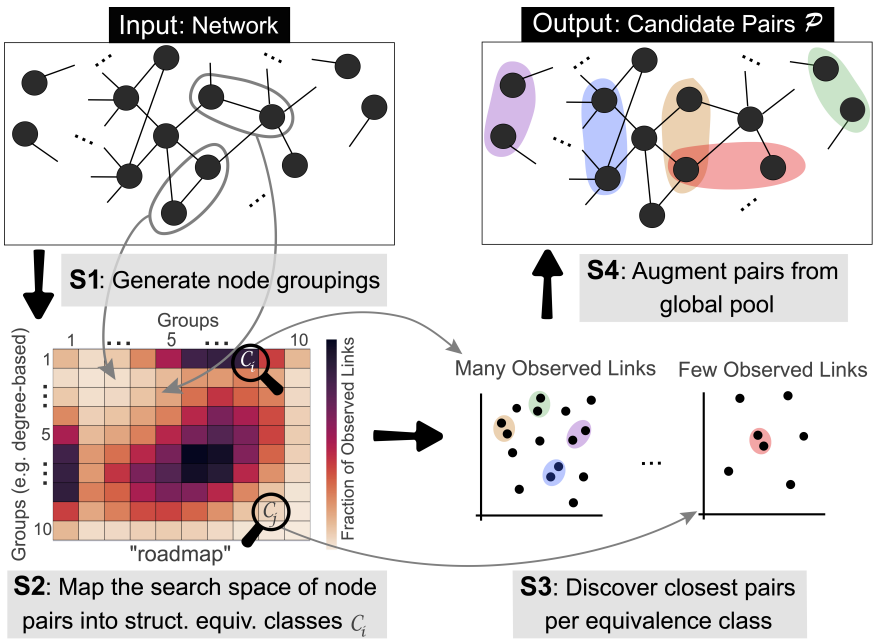


Figure 2.1: Visual representation of LinkWaldo [3]

The method, with the objective to find c unobserved edges, is summarized as:

1. Generate some node embeddings for each node $v \in V$. Suggested embeddings by the authors are NetMF and BINE for bipartite graphs [3].

2. Generate subsets of V using some grouping method. Suggested grouping methods are Log-binned Node Degree (DG), Structural Embedding Clusters (SG), Communities (CG) [3].

3. Construct a Stochastic Block Model matrix, where each column and row correspond to a node grouping or combination of node groupings, and each equivalence class C_{ij} contains the subset of existing edges that can be found between group i and j, which is assumed to reflect the bayesian prior for future edges.

4. Iterate over all classes C_{ij}, perform locality sensitive hashing on all nodes embeddings. This assigns similar nodes to the same bucket with a high probability. Iterate over all possible pairs in each bucket, and score them according to some similarity measure (the authors suggest using the cosine similarity of the node embeddings). Using the Stochastic Block Model value C_{ij} to find an expected number of new edges $\mathbb{E}(|C_{ij} \cap \mathcal{E}_{new}|)$ between i and j, add the closest pairs to a pool $\mathcal{P}$ which corresponds to the algorithm output.

The authors benchmark LinkWaldo against a number of datasets, hiding 20% of the original edges which produces a test graph. The number of output candidate pairs is set to c, where c is manually selected for each individual dataset to be proportional to the number of hidden edges, roughly 10 times greater than n. Computing the resulting recall is done by comparing the number of hidden edges against the c outputted candidates. LinkWaldo is

tested against similarity based methods such as AA and CN, but also Non-Negative Matrix Factorization with Bagging (described in 2.5.2). In their results, LinkWaldo is found to have the strongest recall in 10 out of 13 datasets, and is only marginally outperformed by AA on the remaining three.

Empirically analyzing the time complexity of LinkWaldo, the authors conclude that it *"scales linearly on the number of edges, and subquadratically on the number of nodes"*. More specifically, the time complexity is given by $O(\gamma + |\mathcal{E}| + nb_{max} + c)$, where γ corresponds to the time complexity of grouping nodes and b_{max} is the number of times a node is hashed using LSH.

In the comparison between Non-Negative Matrix Factorization with Bagging against LinkWaldo, it is noteworthy that the authors of LinkWaldo focus on recall rather than precision, with the motivation that *"the returned set of pairs $\mathcal{P}$ does not contain final predictions, but rather pairs for a link prediction method to make final decisions about"*, unlike a *"top-k link prediction, which attempts to predict a small number of links, but misses a large number of missing links in the process, suffering from low recall"*.

Chapter 3

Distributed Approximate Personalized PageRank

As there are currently no published resources on applying PPR to the candidate selection problem, we propose a method for its application. Furthermore, we propose a novel approach to computing an approximate individual PPR for every node in a graph with linear time complexity w.r.t. n that we call Distributed Approximate Personalized PageRank (DAPPR).

The algorithm is of the form $G \times \mathbb{N}^+ \times \omega \mapsto \mathcal{P} \subset V \times V$, taking as input the graph G and an upper bound of the size c of the output set along with some hyper-parameters ω and returns a set of node pair candidates $\mathcal{P}$ deemed likely to form links.

3.1 Applying Personalized PageRank on candidate selection

Computing the stable state PPR for a query node q, as described in section 2.4, yields a probability distribution π_q of randomly walking from q to any node in V. By extracting k nodes from π_q, with the highest probability for each query node q, we can construct the candidate set $\mathcal{P}$. An approach for selecting k would be defining a static k for all query nodes as in the KNN problem. This choice of k would imply that all nodes are expected to be part of equally many unobserved edges. Instead, we assume that the relative degrees of nodes should be preserved. For instance the assumption is that a dangling node with only a single neighbor is likely to be included in fewer unobserved links than a central node of high degree. This Bayesian assumption, similar to the one made in Linkwaldo (see section 2.5.3), entails that the number of output edges per query node q should not be a constant, but rather some function $k(q) \propto |N(q)|$. We suggest the following function:

$$k(q) = \left\lfloor \frac{c}{n} \cdot \frac{|N(q)|}{d} \right\rfloor \tag{3.1}$$

where the first factor represents the average number of candidates to output per node, and the second factor represents the degree of q in relation to the average degree.

This approach causes some discrepancies in the size of the output set $|\mathcal{P}|$ for undirected graphs. Let u be the most probable node in π_v and v be the most probable node in π_u, then the node pairs (u, v) and (v, u) will both be added to $\mathcal{P}$. As the graph is undirected both pairs are considered equal, effectively resulting in $|\mathcal{P}| < c$. Also, equation 3.1 includes a flooring function, which further adds to this issue. To tackle this, another step is added to the final construction of $\mathcal{P}$. When the $k(q)$ most probable node pairs have been added to $\mathcal{P}$ for all q, we continue the set construction by adding the most probable node pairs selected from a pool consisting of all PPR distributions such that $q \in V$ until $|\mathcal{P}| = c$, similar to the approach used in LinkWaldo.

3.2 Personalized PageRank computation

As a single PPR computation through either eigenvalue decomposition, or that of the random surfer approach, both described in section 2.4, output n probabilities, running a unique PPR for each of the n nodes in the graph would require a time complexity of $\Omega(n^2)$. However, since such an approach would perform many redundant computations, the time complexity of generating a unique PPR for each node in the graph could be improved upon significantly. To illustrate this from the perspective of the random surfer approach, assume that a PPR for q (i.e. π_q) is being computed. In the event that the random surfer transitions from q to q's neighbor $v \in N(q)$, all following events given that a teleportation does not occur have the same probability distribution as that of π_v. Thus, simulating these events with a random surfer is redundant if π_v has already been computed.

The concept of the PPR of a node being dependent on its neighbors has been formalized by Widom et. al. in the Decomposition Theorem [14]:

$$\pi_q = (1 - \alpha)e_q + \alpha \frac{1}{|N(q)|} \sum_{v \in N(q)} \pi_v \tag{3.2}$$

The authors also describe the basic dynamic programming algorithm for calculating $\pi_q(t)$ as an approximation of π_q with a proven error $\delta(t)$.

$$\begin{aligned} \pi_q(0) &= e_q \\ \pi_q(t) &= (1 - \alpha)e_q + \alpha \frac{1}{|N(q)|} \sum_{v \in N(q)} \pi_v(t - 1) \\ \delta(t) &= ||\pi_q - \pi_q(t)||_1 = \alpha^t \end{aligned} \tag{3.3}$$

where α is a hyper-parameter in ω in our implementation. By concatenating the results we obtain the $|V| \times |V|$ matrix Π_t.

$$\Pi(t) := \Big[\pi_0(t) \cdots \pi_n(t)\Big]. \tag{3.4}$$

3.3 Approximations

Equations 2.4 and 3.3 offer two different methods for computing the PPR of a node. However, the latter equation provides opportunities for parallelization and intuitive ways to limit the

number of operations and space required while introducing approximations.

As scalability is prioritized in DAPPR, the algorithm makes use of equation 3.3 with some approximations deemed appropriate for a method which aims to output top ranked nodes in π_q. Due to the dependency of each $\pi_q(t)$ on all of its neighbors' $\pi_v(t-1)|v \in N(q)$, the max number of non-zero values held in $\pi_q(t)$ is, in the worst case, increased by a multiple of the maximum node degree for each iteration. This problem can, however, be mitigated by limiting the number of values that are taken into account from each neighbor by some constant, approximating the equation by only selecting the highest probability nodes from each neighbor. In equation 3.5 this operation is represented by the function top_β. In order to ensure that a sufficient number of node candidate pairs can be constructed from the approximate PPR $\hat{\pi}_q(t)$, this ranking needs to include a sufficient number of non-zero entries. Therefore, we suggest using an integer parameter $\lambda \in \mathbb{N}^+$ for selecting the $\beta = \lambda k(q)$ highest probability nodes from each neighbor in $N(q)$ for the computation of $\hat{\pi}_q(t)$. To further limit the growth of the amount of non-zero values, we restrict $\hat{\pi}_q(t)$ so that the maximum number of non-zero values is $\lambda k(q)$.

Using the described modifications of equation 3.3, we formulate the central equations in DAPPR:

$$\begin{aligned}
\hat{\pi}_q(0) &= e_q \\
\hat{\pi}_q(t) &= (1-\alpha)e_q + \alpha \frac{1}{|N(q)|} top_{\lambda * k(q)}\left(\sum_{v \in N(q)} top_{\lambda * k(q)}(\pi_v(t-1)) \right) \\
\hat{\Pi}(t) &:= \left[\hat{\pi}_0(t) \cdots \hat{\pi}_n(t) \right].
\end{aligned} \tag{3.5}$$

In order to solve $\hat{\Pi}(t)$ in equation 3.5, the number of iterations is limited to some number t_{max} which is encountered when a convergence threshold is reached. t_{max} is determined as the largest t such that $\|\hat{\Pi}(t) - \hat{\Pi}(t-1)\|_1 \leq \epsilon$, where ϵ is a hyper-parameter in ω.

3.4 Pseudocode

The pseudocode for DAPPR is presented in algorithm 1. In the pseudocode, Π_{curr} corresponds to $\hat{\Pi}(t)$ in equation 3.5, and Π_{prev} corresponds to $\hat{\Pi}(t-1)$.

Construction of $\mathcal{P}$ is performed according to section 3.1. Note that the for-loop can easily be parallelized.

Algorithm 1 DAPPR

Input: $V, \alpha, \lambda, \epsilon$
Output: Π_{curr}

$\Pi_{prev} \leftarrow I_n$ //Each column q is initialized as e_q.
$\Pi_{curr} \leftarrow 0_{n,n}$ //Initialized as the zero-matrix.

while true **do**
  for $q \in V$ **do** Parallel
    $\Pi_{curr}[:, q] \leftarrow (1-\alpha)e_q + \alpha \frac{1}{|N(q)|} top_{\lambda * k(q)}\Big(\sum_{v \in N(q)} top_{\lambda * k(q)}(\Pi_{prev}[:, v])\Big)$ //Eq. 3.5
  end for
  $\delta \leftarrow \|\Pi_{prev} - \Pi_{curr}\|_1$
  if $\delta < \epsilon$ **then** // Convergence check.
    return Π_{curr}
  end if
  $\Pi_{curr} \leftarrow \Pi_{prev}$
end while

3.5 Complexity Analysis

Running a single iteration of DAPPR will require collecting $\lambda k(q)$ values from d neighbors over all n nodes, where $k(q)$ averages to $\frac{c}{n}$ (according to equation 3.1). The number of iterations, t_{max}, is dependent on the rate of convergence and error threshold ϵ. The final time complexity of DAPPR is $O(nd\frac{c}{n}) = O(dc)$.

To run DAPPR, $\hat{\Pi}(t)$ and $\hat{\Pi}(t-1)$ must be held in memory, containing n rows and up to $\lambda k(q)$ values in each row. Assuming sparse matrices are used, the final space complexity becomes $O(n\frac{c}{n}) = O(c)$.

Chapter 4
Method

The following chapter describes the research questions of this work, and the experimental setup to answer these questions.

4.1 Research Questions

Given the problems statement and aim, stated in 1.1, the thesis will evaluate the following algorithms:

- Distributed Approximate Personalized PageRank
- Breadth First Search
- LinkWaldo
- Nearest Neighbor Descent

on the following research questions (RQs):

- **(RQ1)** How well does the time complexity of each algorithm scale with regards to the number of nodes in the input graph?
- **(RQ2)** To what extent does the output set $\mathcal{P}$ of each algorithm overlap with missing/future edges in an input graph?
- **(RQ3)** How do the results of RQ2 depend on the real-world domain of the input data?

4.2 Methodology

In order to answer the research questions, action research was employed. Action research can be described as a methodology of two parts; situation assessment and problem intervention [8]. These two parts are in turn split into sub parts, where the situation assessment consists of observation and solution, while the problem intervention is made up of evaluation and reflection [8]. The methodology can be viewed as iterative, as it might be repeated until the evaluation is deemed adequate.

4.3 Experimental Setup

The candidate selection algorithms are evaluated on their ability to find edges that have been hidden $\mathcal{E}_{hidden}$ from a graph G. In our experiments $\mathcal{E}_{hidden}$ is the ground truth and consists of 20% of the edges in G. The amount of hidden edges are selected to match the setup used by the authors of LinkWaldo [3] for comparability. Another 20% of edges in G are put in a validation set used by the NN-Descent model (described in 4.3.1). Removing $\mathcal{E}_{hidden}$ from G results in the test graph $G_{test} = (V_{test}, \mathcal{E}_{test})$. For static graphs, the edges to hide are selected uniformly at random. In contrast, for temporal graphs the 20% of edges formed most recently are hidden. This approach may lead to some nodes becoming disjoint in G_{test}, and due to traits shared by all evaluated algorithms, these nodes will not appear as elements of any resulting candidate pairs. Therefore, hidden edges that contain one or more disjoint nodes in G_{test} are removed from $\mathcal{E}_{hidden}$. Consequently, the number of hidden edges is likely to be less than 20% of the number of original edges $|\mathcal{E}|$.

Since the final output $\mathcal{P}$ from each algorithm is not a final link prediction result, but a set of node pairs to perform link prediction on (except for NN-descent which outputs actual predictions), the size of the output set should be larger than or equal to $|\mathcal{E}_{hidden}|$. We evaluate candidate selection algorithms on output sizes $|\mathcal{P}| \leq c = h|\mathcal{E}_{hidden}|$ where $\{h \mid 1 \leq h \leq 10 \ \wedge \ h \in \mathbb{N}\}$. Each algorithm is run only one time for each value of h.

All experiments are conducted on an Intel(R) Xeon(R) CPU E7-8870 v3, 2.10 GHz with 1056 GB RAM.

4.3.1 Algorithms

In order to reduce the search space of possible edges from $O(n^2)$ we consider algorithms of the form $G \times \mathbb{N}^+ \times \omega \mapsto \mathcal{P} \subset V \times V$ which take a graph G and an upper bound of the size of the output set c along with some hyper-parameters ω (unique to each algorithm) as input and return a number of node pair candidates $\mathcal{P}$ deemed likely to form links. The algorithms described below are all implemented in Python 3, using NetworkX [11] for graph management.

Distributed Approximate Personalized PageRank

DAPPR is implemented according to chapter 3. The selected parameters are: $\alpha = 0.8, \lambda = 20, \epsilon = 0.05$, as we have found that these parameters generally yield relatively strong results

on a variety of different graphs. The construction of $\Pi_{curr}[:, q]$ is parallelized over all $q \in V$ in our implementation.

Breadth First Search

Breath First Search (BFS) is a simple graph search algorithm [5] that will be used in the benchmarks as a simpler algorithm to reference the other methods against. Our implementation of BFS on candidate selection works by iterating over all nodes $v \in V$, then for each v the algorithm constructs up to k node pair candidates (u, v) such that $dist(u, v) \leq l$ and $(u, v) \notin \mathcal{E}_{test}$. Specifically, BFS iteratively adds node pairs as candidates layerwise, starting at the set of nodes with the smallest distance to v and continuing with sets of nodes increasingly further away from v until k node pairs have been added. The hyper-parameters ω in BFS are constructed of k and l.

As BFS simply yields k node candidate pairs for each of the n nodes in the graph, where $k = \lfloor \frac{c}{n} \rfloor$, the time complexity of the algorithm is $O(n\frac{c}{n}) = O(c)$.

LinkWaldo

The original implementation of LinkWaldo (described in section 2.5.3) is used, with the source code published by the authors. The algorithm settings are chosen to follow the authors' recommendations [3].

Nearest Neighbor Descent

NN-descent is implemented using the authors' pseudocode with omitted parallelization. As a similarity function, we use a simple logistic regression model that takes the embeddings of two nodes and outputs the score of the node pair (NetMF embeddings are used from the LinkWaldo original implementation [3]). The model is trained on a set of positive and negative training edges extracted from the evaluated test graph. Since the performance of NN-descent is heavily dependent on the similarity function used, the precision of the model on each dataset against the separate validation data is presented in appendix B.1.

We set the hyper-parameters ω within the authors' recommendations, with termination threshold δ is set to 0.001 and a sampling rate is set to $\rho = 0.75$ for fast runtime yet accurate results (the authors tested the accurate $\rho = 1.0$ and the fast $\rho = 0.5$ and found that the fast method still yielded accurate results [7]). k is set dynamically for each iteration over c as $\lfloor \frac{c}{n} \rfloor$, which will in most cases yield fewer candidates than requested, due to the flooring function.

4.3.2 Evaluation metric

The evaluation metrics used should reflect the algorithms' abilities to narrow down the search space, while still outputting node pairs likely to form edges (as stated in the research question). In order to analyze the ability to narrow down the search space, we consider the time complexities of the algorithms. The quality of the candidate node pairs in the algorithm output $\mathcal{P}$ is evaluated through recall.

$$\text{Recall} = \frac{|\mathcal{P} \cap \mathcal{E}_{hidden}|}{|\mathcal{E}_{hidden}|} \tag{4.1}$$

We consider recall a representative measure for the performance of a candidate selection algorithm as it reflects the fraction of hidden edges the algorithm is able to find, while remaining agnostic to the precision. The precision is considered largely irrelevant for these algorithms as downstream link prediction models are expected to filter out false positives from the result. Therefore, a link prediction pipeline using a candidate selection algorithm and a link prediction model should yield high precision and recall while consuming a reasonable amount of computational resources.

4.3.3 Datasets

The algorithms are evaluated on a selection of diverse datasets. The aim is to represent all algorithms' abilities on small and large graphs in a range of different domains, with both static as well as temporal graphs being included. The domains that are selected are *biological* (YEAST, HS-PROTEIN, OGB-DDI), *social* (FACEBOOK1, FACEBOOK2, DIGG, EPINIONS, MATH OVERFLOW, REDDIT), *communication* (ENRON), *citational* (DBLP, ARXIV, OGB-COLLAB), *reviews* (AMAZON), *infrastructure* (US-AIR, POWER, ROUTER, ROADNET-PA). We have chosen a large number of datasets, with large variation, in order to test the versatility of all algorithms and to possibly illuminate if some algorithms perform particularly well on some types of graphs (see Research Question 3). Furthermore, the datasets are collected from a multiple sources in to reduce the risk of overlap or selection bias in the data.

An overview of the datasets is presented in table 4.1.

Name	$\|V\|$	$\|\mathcal{E}\|$	**Temporal**	**Description**
US-AIR [26]	332	2,126		US Air lines
YEAST [3]	2,375	11,693		Proteins in a species of YEAST
FACEBOOK1 [3]	4,039	88,234		Friendships among users
OGB-DDI [13]	4,267	1,334,889		Drug to drug interactions
POWER [26]	4,941	6,594		Electrical grid of western US
ROUTER [26]	5,022	6,258		Router internet
HS-PROTEIN [3]	6,329	147,548		Heat shock protein in humans
DBLP [3]	12,595	49,638		Citation network
ARXIV [3]	18,772	198,110		Co-authorship network of astrophysicists
MATH OVERFLOW [3]	24,818	199,973	✓	Answers to questions on a math forum
FACEBOOK2 [3]	63,731	817,035	✓	Friendships among users
REDDIT [3]	67,180	309,667	✓	Links between forum boards (subreddits)
EPINIONS [3]	75,879	405,740		Users trusting each other's opinions
ENRON [3]	87,273	299,220	✓	Citation network
OGB-COLLAB [13]	235,868	1,024,434		Author collaboration network
DIGG [3]	279,374	1,546,540	✓	Friendships among users
AMAZON [15]	334,863	925,872		Product co-purchasing
ROADNET-PA [15]	1,088,092	1,541,898		Road network of Pennsylvania

Table 4.1: Overview of datasets.

Chapter 5

Results

The following section presents the results of running all benchmarks described in chapter 4.

5.1 Recall

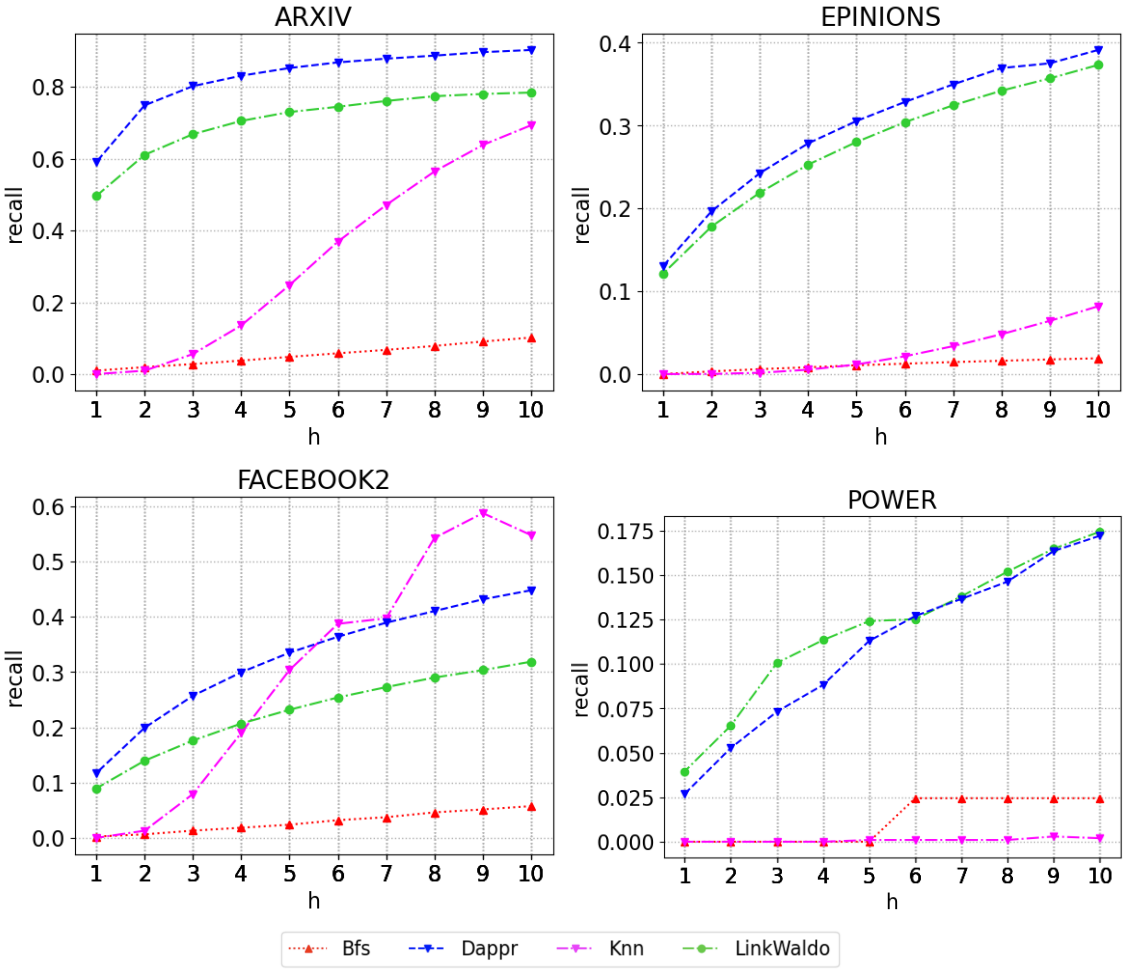


Figure 5.1: Recall for benchmarked algorithms over different output sizes h (where $h|\mathcal{E}_{hidden}| = c$) of a selection of all evaluated datasets (complete results are available in appendix B).

Figure 5.1 shows the recall of the evaluated algorithms for a selection of datasets with different size outputs. The results for all datasets are available in appendix B.

Table 5.1 summarizes the average recall across all output sizes and shows that DAPPR outperforms all other evaluated algorithms in recall in most considered datasets. However, as seen in figure 5.1, DAPPR yields only slightly better recall than LinkWaldo on most datasets and h-values. Finally, BFS does not provide any competitive results.

5.2 Time Complexities

Table 5.2 summarizes the time complexities of the evaluated algorithms. The presented time complexities are motivated in sections 3.3, 4.3.1, 2.5.3 and 2.5.1. The run times for the algorithms on each dataset for two values of h (1 and 10) are presented in appendix B.3.

Dataset	DAPPR	BFS	LinkWaldo	NN-Descent
US_AIR	**0.79**	0.05	0.74	0.39
YEAST	**0.77**	0.05	0.67	0.24
MOVIE_LENS	**0.12**	0.00	0.01	0.07
FACEBOOK1	**0.83**	0.05	0.81	0.70
OGB_DDI	0.80	0.00	**0.89**	0.76
POWER	0.11	0.01	**0.12**	0.00
ROUTER	**0.30**	0.00	0.21	0.00
HS_PROTEIN	**0.95**	0.04	0.89	0.61
DBLP	**0.37**	0.01	0.30	0.07
ARXIV	**0.83**	0.05	0.71	0.32
MATH_OVERFLOW	0.28	0.00	0.26	**0.28**
FACEBOOK2	**0.33**	0.03	0.23	0.31
REDDIT	**0.21**	0.00	0.20	0.01
EPINIONS	**0.30**	0.01	0.28	0.03
ENRON	0.10	0.00	**0.11**	0.01
OGB-COLLAB	0.21	0.00	**0.26**	0.00
DIGG	0.06	0.00	**0.07**	0.01
AMAZON	**0.50**	0.05	0.45	0.00
ROADNET_PA	**0.07**	0.05	0.01	0.00

Table 5.1: Average recall over all output sizes (from figure B.2) per algorithm and per dataset. The best result for each dataset is written in bold text.

Algorithm	Time complexity
DAPPR	$O(dc)$
BFS	$O(c)$
LinkWaldo	$O(\gamma + \lvert\mathcal{E}\rvert + nb_{max} + c)$
NN-descent	$O(c^2/n)$

Table 5.2: Time complexities of the evaluated algorithms.

Chapter 6
Discussion & Conclusion

In this chapter we discuss the results and their implications with regards to the research questions, some limitations of our experiments, and finally suggest recommendations for future work.

6.1 Distributed Approximate Personalized PageRank

RQ1: The time complexity of DAPPR is $O(dc)$, which is significantly lower than that of the exhaustive approach ($O(n^2)$) for many real world graphs where $d \ll n$. While computing the complete PPR for each node would yield a time complexity of $O(n^2)$, DAPPR is able to perform better as only the top ranking nodes in the PPR are considered. Thus, the approximations utilized in DAPPR (as described in section 3.3) result in the algorithm being a considerably efficient method for performing large scale link prediction. With its distributed approach, it is easy to see how DAPPR can be completely parallelized by the number of nodes in the graph, which is a useful property for reducing run time. Finally, it can be concluded that DAPPR is a relatively simple algorithm to implement.

RQ2: Looking at the results in table 5.1, we argue that DAPPR is the best candidate selection algorithm among those evaluated. DAPPR provides the best results on the majority of datasets (13 out of 19 datasets) and is not far from the top performing algorithm in the remaining datasets. Although DAPPR is built on a heuristic approach, it proves to be a versatile method for finding candidate node pairs for link prediction, as it generalizes well to many different types of graphs.

RQ3: Looking at the recall of DAPPR in each dataset, we cannot conclude that its performance is increased or decreased depending on the underlying domain of the dataset.

6.2 Breadth First Search

RQ1: BFS achieves a constant time complexity of $O(c)$ and is thus by far fastest run time of the tested algorithm.

RQ2: With its very simple use of heuristics, BFS is unable to perform at a level that is comparable with the other algorithms on any dataset and it is therefore not possible to conclude that it is a useful algorithm given the results of the experiments.

RQ3: Given the poor overall performance of BFS it is impossible to conclude anything about its domain-specific performance.

6.3 LinkWaldo

RQ1: LinkWaldo presents a sub-quadratic time complexity, with some terms that are dependent on user choices such as grouping method and number of LSH buckets. While LinkWaldo requires the availability of node embeddings, we do not include node embedding generation in the presented time complexity in table 5.2. The reason being that node embeddings are possible to use in later stages of a link prediction pipeline, and therefore node embedding generation might not add to the total amount of computations required in a link prediction pipeline. Assuming simple embeddings and groupings are used, Linkwaldo's can be considered efficient.

Although not mentioned by the authors and not implemented in the source code provided by the authors, parts of LinkWaldo can be parallelized, such as creating groupings, embeddings and applying LSH. Out of the tested methods, LinkWaldo is the most technically demanding algorithm to implement.

RQ2: Based on the resulting recall and time complexity, we consider LinkWaldo to be the second best algorithm across datasets (performing the best in 5 out of 19 datasets). When studying the recall plots in appendix B.2, one can see that LinkWaldo performs generally well, with a recall only slightly lower than that of DAPPR. The plots also show that LinkWaldo follows a similar pattern of recall with different output sizes as DAPPR. With the recommended settings of the authors, LinkWaldo is run with NetMF embeddings on non-bipartite graphs, a type of node embedding that relies on random walks. This, combined with the bayesian prior applied on the Stochastic Block Model which is partially based on log-binned node degrees, may explain some of the similar behavior between DAPPR and LinkWaldo's results.

RQ3: It is not possible to conclude that LinkWaldo's performance is affected by the domain of the datasets.

6.4 Nearest Neighbor Descent

RQ1: The time complexity of NN-Descent presented is sub-quadratic. Although the method is quadratic in relation to c, this number is chosen by the user and is generally significantly smaller than n. Furthermore, the choice of similarity function or link prediction model in

the algorithm might have a large impact on the time complexity. NN-Descent can easily be parallelized, as suggested by the authors.

RQ2: While the recall of NN-Descent is not impressive when compared to DAPPR and LinkWaldo in table 5.1, it achieves strong results on some datasets (with the best recall in 1 out of 19 datasets). Although the underlying assumption of *"a neighbor of a neighbor is also likely to be a neighbor"* could be expected to work particularly well on social graphs (such as the FACEBOOK datasets), it is not obviously superior in any specific category of datasets.

Looking at the plots in figure B.2, it can be observed that NN-Descent performs significantly better with larger output sets. We speculate that the reason for this is that the probability of a node finding a local optimum with a small k is high, and that the probability of replacing the local optima with stronger candidates rises quickly with larger values of k. With this information, it is possible that NN-Descent would have better results for even larger output sets.

Since NN-Descent requires some similarity function to be chosen, our choice and implementation of link prediction model is only one implementation of NN-Descent for this problem. Although the method could behave differently with other similarity functions, our choice of NetMF embeddings on a Logistic Regression model gave high precision scores for the link prediction task (see table B.1).

RQ3: It is not possible to conclude that NN-Descent's performance is affected by the domain of the datasets.

6.5 Limitations and Future Work

Although this thesis explores existing and novel ideas for selecting candidate node pairs for large scale link prediction, the tests were limited to simple graphs with a single node type and edge type. Real world graphs are rarely so simple, and there might be much to gain by taking into account the information of each node's unique attributes and/or being able to utilize edge types. Future work may focus on extending the evaluated algorithms to both take features into account (e.g. NN-Descent's similarity model and LinkWaldo could include these in their embeddings), as well as for predicting types of links. Furthermore, the analysis of the results in this thesis did not take into account structural information of the graphs such as their connectivity.

In the process of developing DAPPR, we observed that the use of a dynamic k largely improved the result. Specifically, k was set according to the bayesian assumption stating that a high degree node is more likely to be part of unobserved edges than a low degree node. We speculate that the same assumption has potential to improve the NN-descent algorithm, and could be investigated in the future. Other than investigating the use of a dynamic k, NN-descent might benefit from alternative initialization methods, such as random walk based initialization as opposed to uniformly random as described in [22]. Furthermore, there are other algorithms for solving the KNN graph problem which could be applied to the node candidate selection problem, such as the ones discussed by Wang et. al. [25].

As the discussion above suggests, there may be some overlap in the process of extracting candidate pairs between DAPPR and LinkWaldo, particularly the random walk. Both of these algorithms perform very well, however, their results differ significantly in compari-

son with those of NN-descent. The results show that the random walk aided algorithms are sometimes outperformed by NN-descent, indicating that the different types of algorithms might have different strengths. In order to try combining the strengths of DAPPR and NN-descent, it might be relevant to include the dependency of a trained link prediction model in DAPPR. Specifically, it could be used in the *top* function used in equation 3.5, with the goal of utilizing the information provided by the link prediction model in order to determine what entries *top* should return. Thus, the link prediction model would influence the approximations of DAPPR, possibly improving the results of the algorithm.

6.6 Conclusion

In this thesis, we focus on the problem of mitigating the issue of a large search space in the link prediction task for large graphs. A novel algorithm for the task, DAPPR, is presented and evaluated in terms of performance along with three different algorithms across a wide range of datasets.

We conclude that DAPPR is the best performing algorithm among those evaluated and simultaneously has a relatively simple implementation. DAPPR generalizes well across different types of graphs including social, communication, and biological graphs. The algorithm is closely followed in terms of performance and generalization by LinkWaldo. While NN-descent is many times outperformed by both DAPPR and LinkWaldo, it presents strengths on social graphs and is especially strong when the output set is large. Finally, results show that a simple BFS algorithm is not sufficient to solve the problem of node candidate selection for link prediction.

References

[1] Lada Adamic and Eytan Adar. Friends and neighbors on the web. *Social Networks*, 25:211–230, 07 2003.

[2] Asan Agibetov. Neural graph embeddings as explicit low-rank matrix factorization for link prediction. *Pattern Recognition*, 133:108977, 2023.

[3] Caleb Belth, Alican Büyükçakir, and Danai Koutra. A hidden challenge of link prediction: Which pairs to check? *2020 IEEE International Conference on Data Mining (ICDM)*, pages 831–840, 2020.

[4] A. Bickle. *Fundamentals of Graph Theory*. Pure and Applied Undergraduate Texts. American Mathematical Society, 2020.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[6] DB-engines. DBMS popularity broken down by database model. `https://db-engines.com/en/ranking_categories`, accessed: 2022-11-02, 11 2022. accessed: 2022-11-02.

[7] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. *Proceedings of the 20th International Conference on World Wide Web*, page 577–586, 2011.

[8] Paulo Sergio Medeiros dos Santos and Guilherme Horta Travassos. Action research can swing the balance in experimental software engineering. In *Advances in Computers*, pages 205–276. Elsevier, 2011.

[9] Liang Duan, Shuai Ma, Charu Aggarwal, Tiejun Ma, and Jinpeng Huai. An ensemble approach to link prediction. *IEEE Transactions on Knowledge and Data Engineering*, 29(11):2402–2416, 2017.

[10] David F. Gleich. Pagerank beyond the web. *SIAM Review*, 57(3):321–363, 2015.

[11] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, 2008.

[12] William L. Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *CoRR*, abs/1709.05584, 2017.

[13] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

[14] Glen Jeh and Jennifer Widom. Scaling personalized web search. *Proceedings of the 12th International Conference on World Wide Web*, pages 271–279, 2003.

[15] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[16] David Liben-nowell and Jon Kleinberg. The link prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58, 01 2003.

[17] Ryan N. Lichtenwalter, Jake T. Lussier, and Nitesh V. Chawla. New perspectives and methods in link prediction. *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 243–252, 2010.

[18] Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. A survey of link prediction in complex networks. *ACM Comput. Surv.*, 49(4), dec 2016.

[19] Afraz Jaffri Merv Adrian. Market guide for graph database management systems. Technical report, Gartner, August 2022. `https://www.gartner.com/doc/reprints?id=1-2B27WUOY&ct=220908&st=sb&submissionGuid=99861651-015f-4819-90e1-9ead8cbaa414`, accessed: 2022-10-31.

[20] Gleich Nassar, Benson. Neighborhood and pagerank methods for pairwise link prediction. *Social Network Analysis and Mining*, 2020.

[21] Neo4j. Impossible is nothing: The history (& future) of graph data [graphconnect recap]. `https://neo4j.com/blog/history-and-future-of-graph-data/`, Aug 2015. accessed: 2022-11-02.

[22] Neo4j. The neo4j graph data science library manual v2.2: Applying a trained model for prediction. `https://neo4j.com/docs/graph-data-science/2.2/machine-learning/linkprediction-pipelines/predict/`, Aug 2022. accessed: 2022-11-17.

[23] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[24] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *CoRR*, abs/1408.2927, 2014.

[25] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. 2021.

[26] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *Advances in Neural Information Processing Systems*, 31, 2018.

[27] Yinzuo Zhou, Chencheng Wu, and Lulu Tan. Biased random walk with restart for link prediction with graph embedding method. *Physica A: Statistical Mechanics and its Applications*, 570:125783, 2021.

[28] Marinka Zitnik, Monica Agrawal, and Jure Leskovec. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics*, 34(13):i457–i466, 06 2018.

Appendices

Appendix A
Notation

A.1 Mathematical Symbols

Notation	Description
$G = (V, \mathcal{E})$	Graph, Nodes, Edges
$\|V\| = n$	Number of nodes
$N(v)$	The set of neighboring nodes of node v
d	The average degree of all nodes in the graph
$dist(u, v)$	Length of the shortest path between the nodes u and v
A	Adjacency matrix
α	Non-teleportation probability
k	Number of output candidate pairs per node
$\mathcal{P}$	The set of candidate pairs
ω	Individual hyper-parameters for a given algorithm

Table A.1: List of major mathematical symbols.

Appendix B
Complete Results

B.1 Nearest Neighbor Descent Classifier Precision

Dataset	Model Precision
US_Air	0.75
YEAST	0.94
Movie_Lens	0.80
FACEBOOK1	0.94
OGB_DDI	0.78
Power	0.99
Router	0.95
HS_Protein	0.96
DBLP	0.94
ARXIV	0.95
MATH_Overflow	0.80
FACEBOOK2	0.91
REDDIT	0.81
EPINIONS	0.82
ENRON	0.91
OGB-COLLAB	0.97
DIGG	0.82
AMAZON	0.97
ROADNET_PA	0.99

Table B.1: Precision on validation set for the Logistic Regression model used in NN-Descent for each dataset

B.2 Recall

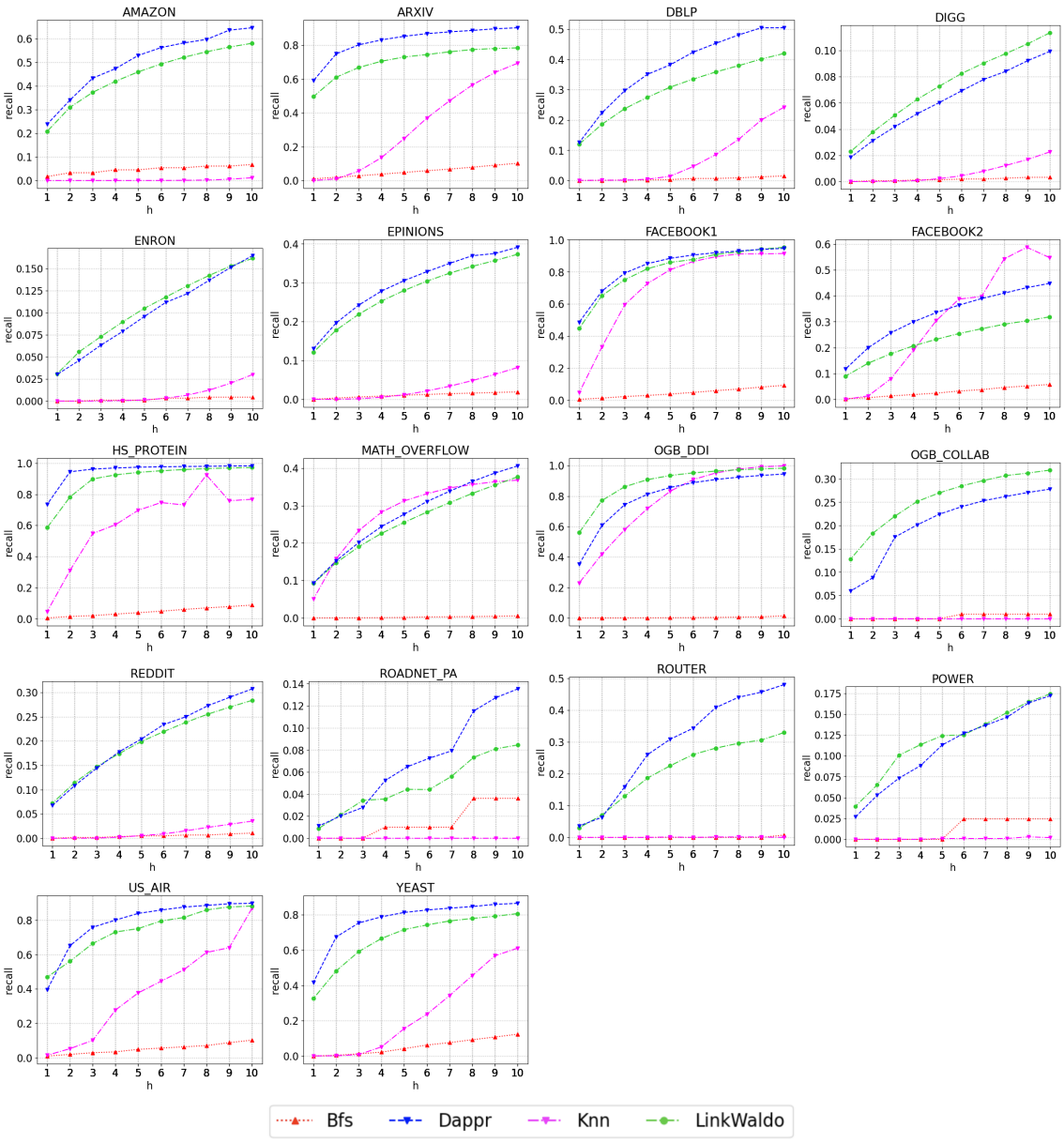


Figure B.1: Recall for benchmarked algorithms over different h (where $\mathcal{E}_{hidden} * h = c$)

B.3 Time

Dataset	DAPPR		BFS		LinkWaldo		NN-Descent	
	h = 1	h = 10	h = 1	h = 10	h = 1	h = 10	h = 1	h = 10
US_AIR	0.4	1.1	0.0	0.0	1.5	1.1	0.5	1.2
YEAST	3.8	7.8	0.0	0.0	17.4	7.0	0.6	7.1
MOVIE_LENS	2.0	13.7	0.0	0.1	13.7	17.9	7.1	42.2
FACEBOOK1	12.6	20.7	0.1	0.4	54.8	48.2	10.4	62.7
OGB_DDI	123.4	324.5	0.8	8.8	507.8	764.3	188.8	3360.9
POWER	11.7	8.9	0.0	0.0	22.4	10.0	0.6	6.4
ROUTER	15.2	15.9	0.0	0.0	25.4	9.3	0.8	5.5
HS_PROTEIN	19.6	43.0	0.1	0.5	90.4	37.2	21.7	126.0
DBLP	11.8	19.1	0.1	0.2	45.5	47.1	11.2	45.9
ARXIV	26.0	70.5	0.2	0.7	88.8	105.3	58.8	189.5
MATH_OVERFLOW	18.0	50.1	1.3	1.1	60.1	61.2	87.2	1027.1
FACEBOOK2	47.5	266.4	1.7	5.8	746.0	710.3	401.2	1001.9
REDDIT	28.8	135.2	1.2	8.7	788.4	819.2	258.3	501.9
EPINIONS	90.6	538.0	2.2	5.0	645.3	678.3	773.4	1037.4
ENRON	16.4	64.1	4.2	4.3	491.1	479.8	322.0	631.4
OGB_COLLAB	28.5	70.9	3.6	3.9	10859.4	10546.3	1245.8	6277.7
DIGG	75.2	410.6	6.0	39.6	6090.0	5537.4	4936.7	5830.4
AMAZON	247.4	1112.8	4.9	11.3	3084.5	3409.2	6200.3	12856.5
ROADNET_PA	382.3	1066.5	11.6	24.6	10583.3	7137.2	33698.3	119546.6

Table B.2: Time in seconds for running algorithms on respective datasets, with h equal to 1 and 10 of a single run. DAPPR and BFS were run with parallelization, while NN-Descent and LinkWaldo were not. The run times should not be compared between algorithms, however can be used for analyzing how an individual method scales.

Appendix C

Individual Author Contributions

This thesis and the work behind it has been done mainly in a highly collaborative manner. However, the largest ownership of each section of the thesis is specified in table C.1.

The implementation work was somewhat divided, as Mahir Hambiralovic mainly owned the running and monitoring of the benchmarks along with the implementations of the specific candidate selection algorithms. Meanwhile, Filip Kalkan owned the implementation of the benchmarking framework and data processing.

Section	Author
Problem Statement	Mahir Hambiralovic
Contribution	Mahir Hambiralovic
Preliminaries	Filip Kalkan
Node Embedding	Filip Kalkan
Link Prediction Models	Mahir Hambiralovic
Locality Sensitive Hashing	Filip Kalkan
Personalized PageRank	Mahir Hambiralovic
Related Work	Mahir Hambiralovic
Applying Personalized Pagerank on Candidate Selection	Filip Kalkan
Personalized PageRank Computation	Filip Kalkan
Approximations	Filip Kalkan
Pseudocode	Filip Kalkan
Complexity Analysis	Mahir Hambiralovic
Setup	Mahir Hambiralovic
Algorithms	Filip Kalkan
Evaluation Metric	Mahir Hambiralovic
Datasets	Mahir Hambiralovic
Recall	Mahir Hambiralovic
Time Complexities	Filip Kalkan
DAPPR	Mahir Hambiralovic
Breadth First Search	Mahir Hambiralovic
LinkWaldo	Filip Kalkan
NN-descent	Filip Kalkan
Limitations and Future Work	Mahir Hambiralovic
Conclusion	Filip Kalkan

Table C.1: Individual contributions.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2023-01-20

EXAMENSARBETE Finding Candidate Node Pairs for Link Prediction at Scale
STUDENTER Filip Kalkan, Mahir Hambiralovic
HANDLEDARE Per Andersson (LTH)
EXAMINATOR Alma Orucevic-Alagic (LTH)

Länkförslag i Enorma Nätverk

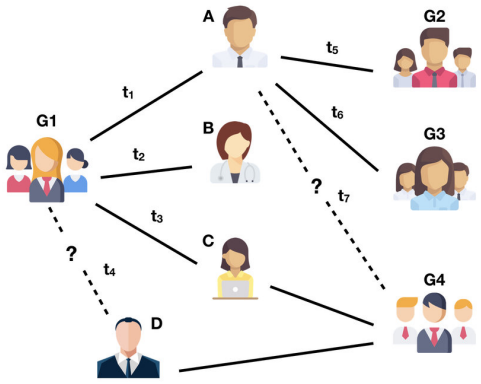
POPULÄRVETENSKAPLIG SAMMANFATTNING **Filip Kalkan, Mahir Hambiralovic**

Att få förslag för personer man kanske känner, filmer man skulle kunna tycka om eller produkter man kanske är intresserad av är ingen ovanlig företeelse idag. Men att dessa rekommendationer ofta blir dåliga ganska fort är inte heller ovanligt och visar på behovet av avancerade lösningar för att hitta dolda länkar i ett hav av information.

I vår masteruppsats har vi undersökt problemet med att hitta lämpliga nodpar för länkprediktion. Med andra ord behandlar vår uppsats frågan "Om vi bara får utvärdera ett begränsat antal nodpar, vilka nodpar bör vi utvärdera för att se om de borde vara sammanlänkade eller inte?". Detta är en viktig fråga inom områden som sociala medier och rekommendationssystem, då det kan hjälpa till för att tillämpa länkprediktion på enorma grafer där det hade tagit för lång tid att utvädera alla möjliga nodpar, och således leda till mer relevanta rekommendationer.

Vi har utvecklat en ny algoritm, baserad på teknik som används i Googles sökmotor, för att lösa detta problem, som vi kallar Distributed Approximate Personalized PageRank (DAPPR). I vår uppsats har vi jämfört DAPPR med andra algoritmer genom att använda olika mätmetoder och testdata, och resultaten visar tydligt att DAPPR presterar bättre än de andra algoritmerna.

Så hur fungerar DAPPR? I grunden använder algoritmen nätverksstruktur för att avgöra vilka noder som är viktiga för varandra. Mer specifikt kollar algoritmen på varje nod (utgångsnod), och tar reda på vilka andra noder som är viktigast för utgångsnoden. Detta görs genom att konceptuellt vandra runt i grafen, från nod till nod via länkar, och samtidigt se till att besöka utgångsnoden ofta. När många vandringssteg har utförts, kollar algoritmen på vilka noder som oftast besöktes under vandringen, och utser dessa till viktiga noder i relation till utgångsnoden. Algoritmen rekommenderar sedan att undersöka huruvida ett antal par av noder, bestående av utgångsnoder och deras viktigaste noder, bör vara sammanlänkade.



Vi har undersökt möjligheterna att använda DAPPR i olika typer av nätverk, och resultaten visar att algoritment är flexibel och välfungerande i många olika sammanhang. Flexibiliteten tillsammans med de starka resultaten indikerar att algoritmen har potential att bli ett viktigt verktyg för att möjliggöra bra rekommendationer i enorma nätverk.