Matching Models for Graph Retrieval

Author: Yash Jain
yashjain@cse.iitb.ac.in
Chitrak Gupta
baekgupta@cse.iitb.ac.in

Supervisors:
Prof. Soumen Chakrabarti
soumen@cse.iitb.ac.in
Prof. Abir De
abir@cse.iitb.ac.in
Abstract

Graph Retrieval has witnessed continued interest and progress in the past few years. In this report, we focus on neural network based approaches for Graph matching and retrieving similar graphs from a corpus of graphs. We explore methods which can soft predict the similarity between two graphs. Later, we gauge the power of a particular baseline (Shortest Path Kernel) and try to model it in our product graph random walks setting while making it more generalised.
# Contents

1 Introduction 4
  1.1 Objectives ........................................... 4
  1.2 Contributions ....................................... 4

2 Matching Models for Graph Retrieval via Deep Learning 5
  2.1 Introduction ......................................... 5
  2.2 Notation ........................................... 5
  2.3 Method ............................................... 6
  2.4 Results ............................................. 7

3 Datasets 8
  3.1 SQuAD [6] ........................................... 8
  3.2 COX2 .................................................. 8
  3.3 PTC ................................................... 8
  3.4 MUTAG ............................................... 8
  3.5 Clevr VQA ........................................... 8
  3.6 Pascal VOC Keypoints .................................. 9

4 Baselines 10
  4.1 Non-trainable Heuristics based ......................... 10
    4.1.1 Random Prediction Baseline ....................... 10
    4.1.2 Graph Kernels ................................... 10
    4.1.3 Shortest Path Kernel [1] ....................... 10
    4.1.4 Reweighted Random Walk kernel [3] ............. 11
  4.2 Trainable Methods ................................... 11
    4.2.1 GMN (Graph Matching Networks) [5] ........... 11

5 Biased/Trainable walks in GxNet 12
  5.1 Introduction ......................................... 12
  5.2 Method ............................................... 12
  5.3 Results ............................................. 13

6 Neural matching with Shortest path product graph 15
  6.1 Introduction ......................................... 15
  6.2 Method ............................................... 15
  6.3 Results ............................................. 16
  6.4 Conclusion .......................................... 16
List of Figures

4.1 Left: Siamese network using graph embedding method, Right: GMN model . . . 11

List of Tables

2.1 Mean Average Precision Values across various baselines. . . . . . . . . . . . . 7

5.1 Mean Average Precision values of various approaches. The best performing ap-
proaches are emphasised, suggesting superior performance of GxNET. GMN (Graph
Matching Networks), RRW (Reweighted Random walk kernel) SPK (shortest path
kernel) $\phi$ refers to the trainable parameters of learning transition probabilities in
the random walks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

6.1 Mean Average Precision values. The best performing approaches are emphasised,
suggesting the superior performance of GMN. Note that GMN (Graph Matching
Networks), RW (Random walk kernel) SPK (shortest path kernel), SPRW (Short-
est Path Random Walk) has three settings (a) unbiased random walks (b) biased
random walks with $\alpha = 1$ and (c) biased random walks with $\alpha = inf$ . . . . . . 16
Chapter 1

Introduction

1.1 Objectives

The objective of this study was to:

- Improve the performance of the SIGIR paper "Deep Neural Matching Models for Graph Retrieval" [4] by incorporating trainable random walks in GxNet
- Implement new graph kernel baselines apart from the GxNet paper [4]
- Survey new graph datasets for robust testing
- Develop novel shortest path product graph approach to generalise Shortest path kernel [2] in random walk setting

1.2 Contributions

During this study, we developed a thorough understanding of graph based information retrieval systems. We learnt the pitfalls in various graphs based IR systems and tried to improve the performance of GxNet by incorporating trainable random walks in its architecture. Further, we tried to generalise the Shortest Path Kernel approach by integrating its ideas into product graph regime to match its performance with Shortest Path Kernel in random walk framework on several datasets.
Chapter 2

Matching Models for Graph Retrieval via Deep Learning

2.1 Introduction

The graph retrieval problem is to return ‘relevant’ or ‘good’ response graphs from a corpus, given a query graph. The solution to this problem requires scoring (and thereby rank) the corpus graphs with respect to a query graph using some technique. Though in each dataset, the concept of relevance is different, i.e. either semantic or structural and hence, a single heuristic will not work for both types of such datasets. Most prevalent graph kernels quantify the structural similarity between graphs, providing high similarity scores to isomorphic graphs. That’s why this check of similarity between graphs is also called isomorphism check. On the other hand, in many other applications like drug active-sites identification or question answering from documents, we may want a corpus graph to be prompted relevant if a subgraph of the corpus graph is isomorphic to the query graph. Thus, GxNet proposes an E2E supervised approach which implicitly learns such isomorphic subgraphs.

2.2 Notation

Given a set of query graphs \( Q = \{G_q = (V_q, E_q, F_q, T_q)\} \) and a set of corpus graphs \( C = \{G_c = (V_c, E_c, F_c, T_c)\} \). Each query graph \( G_q \) is associated with a set of good (relevant) corpus graphs

\[
C_{q+} = \{G_c \in C | \text{Relevance}(G_q, G_c) = 1\} \tag{2.1}
\]

and a set of bad (irrelevant) corpus graphs

\[
C_{q-} = \{G_c \in C | \text{Relevance}(G_q, G_c) = 0\}. \tag{2.2}
\]

For each query or corpus graph \( G \bullet \), we also observe a set of node features \( F \bullet = \{f_u\}_{u \in V \bullet}\) and a set of edge types \( T \bullet = \{t_e\}_{e \in E \bullet}\), where \( \bullet \) may be one of \( q \) and \( c \). The meaning of the underlying node features and edge types vary across datasets.

**Product graph.** For each pair of query \( G_q \) and corpus graph \( G_c \), we construct the tensor product graph \( G^{qc} = G_q \times G_c \):

\[
G^{qc} = G_q \star G_c = (V^{qc}, E^{qc}, F^{qc}, T^{qc}), \quad \text{where}
\]

\[
V^{qc} = V_q \times V_c, \quad E^{qc} = E_q \times E_c, \quad F^{qc} = F_q \times F_c, \quad T^{qc} = T_q \times T_c. \tag{2.3}
\]

Edges are included in the product graph \( G^{qc} \) according to the rule

\[
((u, u'), (v, v')) \in E^{qc} \iff (u, v) \in E_q \text{ and } (u', v') \in E_c; \tag{2.4}
\]
the node feature for a node \((u, u')\) in \(G^{qc}\) are set as follows:

\[
 f(u, u') = (f_u, f_{u'}) \forall u \in G_q, u' \in G_c;
\]  

(2.5) and the edge labels in \(G^{qc}\) are set as follows:

\[
 t(u, u'), (u, v), (v, v') \text{ if } (u, v) \in E_q \text{ and } (u', v') \in E_c
\]  

(2.6)

### 2.3 Method

GxNet computes an embedding \(z_{u, u'}\) for each node \((u, u')\) in the product graph \(G^{qc}\). In addition, GxNet also computes an embedding \(r_{ee'}\) for each edge in the product graph based on the edge features \(t_e, t_{e'}\). More specifically, we have:

\[
 z_{u, u'} = g_{node}(f_u, f_{u'}) \quad (2.7)
\]

\[
 r_{e, e'} = g_{edge}(t_e, t_{e'}) \quad (2.8)
\]

In the next step, GxNet constructs the embedding vector for each \((source\ node, edge, target\ node)\) triple in the product graph \(G^{qc}\), denoted \(\tau = ((u, u') \xrightarrow{e} (v, v'))\), as \(h_{\tau} = \rho(z_{u, u'}, r_{e, e'}, z_{v, v'})\),

(2.9) which in turn gives us a score or scalar weight of \(\tau\), given by

\[
s_{\tau} = \sigma(W_{s, \theta} h_{\tau}), \quad (2.11)
\]

where \(s_{\tau}\) indicates an affinity score for the edge \(\tau\). Here \(g_{node}, g_{edge}, \rho, \sigma\) are suitable standard networks with appropriate non-linearities. The parameter \(\theta\) stands for the set of all the trainable parameters in these networks. The final score \(y_{\theta}(G_q, G_c)\) is calculated by aggregating the scores \(s_{\tau}\) of a Subgraph \(S\) in the product graph \(G_{qc}\). The idea is that if \(G_q\) and \(G_c\) are positively related then there will be a subgraph in both of them (and subsequently \(G_{qc}\)) which highly ‘matches’ with each other. The neural networks hopes to learn this latent substructure.

Since finding such a \(S\) is tedious and almost impractical GxNet employs Random walks of length \(K\) (a hyperparameter) to sample a part of the product graph and get an aggregated score with the hope of sampling the similar subgraph in \(G_q\) and \(G_c\). Formally, for each \(k \in [K]\), it performs \(n\) random walks with restarts in the product graph. The start probability of all nodes is kept equal. At any node, transition probabilities for all edges are equal. After sampling a random walk, it samples the subgraph as the induced subgraph of a random walk. After sampling subgraphs \(S_i, i \leq n\) computes the approximate maximum of the aggregated affinity scores in the following way.

\[
score_k(G_q, G_c) = \sum_{i \in [n]} \frac{\sum_{\tau \in S_i| |S_i| = k} s_{\tau} \exp(\sum_{\tau \in S_i| |S_i| = k} s_{\tau})}{\sum_{i \leq n} \exp(\sum_{\tau \in S_i| |S_i| = k} s_{\tau})} \quad (2.12)
\]

Finally, GxNet compute the average affinity over the different lengths of the walk, penalized by the length of path \(k\). Hence, the final confidence score of the relevance of \(G_q\) with \(G_c\) is

\[
y_{\theta}(G_q, G_c) = \frac{1}{K} \sum_{k \in [K]} score_k(G_q, G_c), \quad (2.13)
\]

To train the parameter \(\theta\), GxNet uses pairwise ranking loss function

\[
 \arg\min_{\theta} \sum_{q \in Q} \sum_{c \in C_q, G'_{c'} \in C_{q'}} \text{ReLU}\left(\Delta + y_{\theta}(G_q, G_c) - y_{\theta}(G'_q, G'_{c'})\right) \quad (2.14)
\]

where \(\Delta\) is a tuned margin and \(\text{ReLU}\) is \(\max\{\cdot, 0\}\).
Table 2.1: Mean Average Precision Values across various baselines.

| Dataset      | GxNet | GMN | RRW |
|--------------|-------|-----|-----|
| VQA (Clevr)  | 0.98  | 0.38| 0.65|
| SQuAD        | 0.34  | 0.27| 0.37|
| COX2         | 0.31  | 0.35| 0.37|

2.4 Results

GxNet was evaluated on two datasets, i.e. VQA and SQuAD, and was compared against the baselines GMN and RRWM. Please refer to the table 2.1 for detailed results.

Preliminary experiments suggested that GxNet showed great promise in graph neural matching and was highly resilient to noise in the data compared to its counterparts. We will delve in further about GxNet in future chapters.
Chapter 3

Datasets

3.1 SQuAD [6]

Stanford Question Answering Dataset (SQuAD) is a reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable. To make it amiable for the graph retrieval task, each natural language sentence was converted into its dependency parse using spacy dependency parser. The corpus graph was made up of the rest of the spans from the reading passage.

3.2 COX2

COX2 is a molecular dataset where every graph is classified into one of two classes.

3.3 PTC

PTC is a molecular dataset where molecules are classified according to their carcinogenicity on rodents. This dataset is classified into 4 datasets PTC_FM, PTC_FR, PTC_MM and PTC_MR where M/F stands for male/female and M/R stands for mice/rodent.

3.4 MUTAG

This dataset is a molecular dataset where molecules are classified according to their mutagenicity on bacteria.

3.5 Clevr VQA

The Clevr VQA benchmark consists of an image corpus where each image is associated with a scene graph. The nodes of a scene graph have attributes concerning shape, color, texture and size. Edges of a scene graph represent spatial relations like ‘left of’ and ‘behind’. Clevr queries are originally in natural language and usually parsed into a graph. The query graphs were generated by placing five objects randomly on the scene. Object attributes (shape, color, size, material) are sampled uniformly randomly from predefined choices. Next, spatial relations (left, right, front, behind) are computed between object pairs. The label of an edge is a 1-hot vector of length 4, corresponding to the above relations. Positive corpus graphs are generated by adding more objects to the scene whose attributes are decided uniformly randomly, and edges are added by computing relationships between object pairs as before. Negative corpus graphs are sampled from the positive samples of other queries since the probability of a corpus graph
matching two query graphs well is very low. To test the resilience of various algorithms, noise is added to the color \((\text{RGB} \in \mathbb{R}^3)\) and size \((\in \mathbb{R})\) node attributes. Each component is in \([0, 1]\), to which an independent Gaussian noise with varying standard deviation was added.

### 3.6 Pascal VOC Keypoints

Pascal Visual Object Classification challenge consist of 20 image classes, where each image was parsed into an image graph using keypoints as nodes. The nodes signify the VGG16 features of the bounding box of the part of image local to the keypoint. The dataset was later rejected for the experiments as the node features within the classes does not make sense semantically, i.e., nodes of an image-graph of aeroplane cannot be compared to the nodes of a car but only be compared to an another instance of aeroplane image-graph. This was unsuitable for our proposed task of deep graph matching with a query and a set of corpus graphs.
Chapter 4

Baselines

4.1 Non-trainable Heuristics based

4.1.1 Random Prediction Baseline

Here we arbitrarily predict the similarity between the graphs.

4.1.2 Graph Kernels

A kernel is a function that takes two vectors and returns the similarity between these two vectors as the dot product between some high dimensional mapping of the two vectors. The two conditions equivalent to a kernel function are symmetricity and positive semi-definite property. In context of graph kernels, the higher dimensional feature space most often corresponds to the decomposition of the graph into some specific substructures like walks, paths, sub-graphs which may or may not be further refined. Here we consider two different graph kernels: reweighted random walk kernel and shortest path kernel.

4.1.3 Shortest Path Kernel [1]

This kernel essentially calculates the similarity between the graphs by comparing the two graphs on the basis of the no of shortest paths of some specific length and connecting two nodes of some specific node labels. Formally let \( \phi(n, l_u, l_v)(G) \) denote the number of shortest paths in graph \( G \) of length \( n \) that exist between two nodes with node labels \( l_u \) and \( l_v \). Then we have,

\[
\phi(n, l_u, l_v)(G) = \left| \{ (u, v) : \text{label}(u) = l_u, \text{label}(v) = l_v \text{ shortest path}(u, v) = n \ \forall u, v \in V_G \} \right|
\]  

(4.1)

We do this for all such possible \( k, l_u \) and \( l_v \) which can be done in polynomial time thanks to algorithms like Floyd-warshall. Once done this, then to compute the similarity of some graph in the corpus \( G_c \) with that a query graph \( G_q \) one can do the following.

\[
k_{\text{shortest path}}(G_q, G_c) = \sum_{(n, l_u, l_v)} \phi_{(n, l_u, l_v)}(G_q) \cdot \phi_{(n, l_u, l_v)}(G_c)
\]  

(4.2)

One can rewrite this formula as the dot product between two vectors, each representing the high dimensional vector mapping of graphs.

\[
k_{\text{shortest path}}(G_q, G_c) = \Phi(G_q) \cdot \Phi(G_c)
\]  

(4.3)

Normalised shortest path kernel[7]: One can normalise the vectors \( \Phi(G) \) and obtain the normalised shortest path kernel values.

\[
k_{\text{Norm_shortest path}}(G_q, G_c) = \frac{\Phi(G_q) \cdot \Phi(G_c)}{||\Phi(G_q)||_2 \cdot ||\Phi(G_c)||_2}
\]  

(4.4)
4.1.4 Reweighted Random Walk kernel [3]

The main idea in a random walk kernel for graph similarity is to count the number of matching walks between the two graphs where the nodes are matched on the basis of their node labels. But one can obtain this kernel value in the by using the following formulation.

\[ k_{\text{random \_ walk}}(G_q, C_c) = \sum_{i,j=1}^{\left|V_x\right|} \sum_{n=0}^{N} \lambda_n A_{x}^{n}|_{ij} \]  \hspace{1cm} (4.5)

But they are not effective in real-life datasets when there are a lot of noisy and outlier nodes present in the association graph of two similar graphs. To solve this, a reweighted random walk kernel iteratively updates the confidence in the candidate correspondences between nodes of two graphs, hopefully becoming robust against noisy and outlier nodes.

4.2 Trainable Methods

4.2.1 GMN (Graph Matching Networks) [5]

This method outputs the similarity score by using a cross graph attention mechanism. This is in contrast to siamese networks where the two graphs are independently mapped to two vectors and then compared. In GMN, the information of how similar or dis-similar a node in one graph is with every other node in the other graph is also used while calculating the node embeddings.

Figure 4.1: Left: Siamese network using graph embedding method, Right: GMN model
Chapter 5

Biased/ Trainable walks in GxNet

5.1 Introduction

The SIGIR GxNet paper was based on Uniform Random walks to sample the subgraph within the product graph $G_{qc}$. In a scenario where the product graph is very huge there might be a situation where the uniform random walk completely misses the target subgraph resulting in a low score and thereby poor ranking. To avoid such a situation, we thought of training the walk paths instead of doing a random sampling. We changed the start probability and transition probabilities to depend on node or edge features in simple parametric ways. Consider the edge $(u, u') \rightarrow (v, v')$ in the product graph connecting $(u, u')$ to $(v, v')$. The corresponding node and edge features are $f_{(u, u')}, t_{(u, u')}, (v, v')$, and $f_{(v, v')}$. For random walk over the product graph, we designate the weight of the edge as $g_\phi(f_{(u, u')}, t_{(u, u')}, (v, v'), f_{(v, v')})$. where $g_\phi$ is a neural network denoting a new set of non-linear parameters $\phi$. Edge weights are then suitably scaled per-node to make the walk probability matrix stochastic.

5.2 Method

Following the notation of GxNet, in the trainable walks the weight of the edge is trained over a neural network whose parameters are denoted by $\phi$. Thus, in the trainable walks the pairwise ranking loss function would become:

$$
\mathcal{L}(G, C, C'; \theta, \phi) = \sum_{G_q} \sum_{G_c, G_c'} \text{ReLU}(\Delta + y_{\theta, \phi}(G_q, G_c') - y_{\theta, \phi}(G_q, G_c))
$$

(5.1)

We break down the training process in three possible ways:

- Train both $\theta$ and $\phi$ simultaneously
- Train $\theta$ and $\phi$ alternatively
- Train $\phi$ first and then train $\theta$ while keeping the parameters of $\phi$ fixed

Along with the training procedure, we also experimented with the score aggregation function for a random walk across two formulations:
• MAX formulation: Selecting the max score across the $n$ walks of $k \epsilon [K]$ lengths

$$y_{\theta, \phi}(G_q, G_c, k) = \max_{S \sim P_\phi(S|G^qc)} (y_\theta(G_q, G_c, S))$$

$$\approx \frac{1}{\alpha} \log \left( \sum_{S \sim P_\phi(S|G^qc)} \exp(\alpha y_\theta(G_q, G_c, S)) \right)$$

$$= \frac{1}{\alpha} \log \left( N \times \sum_{S|S|=k} \exp(\alpha \times y_\theta(G_q, G_c, S)) \frac{P_\phi(S|G^qc)}{P_{\text{BiasedWalk}}(S|G^qc)} \ast P_{\text{BiasedWalk}}(S|G^qc) \right)$$

$$= \frac{1}{\alpha} \log \left( \sum_{S \sim \text{BiasedWalk}|S|=k} \exp(\alpha \times y_\theta(G_q, G_c, S)) \frac{P_\phi(S|G^qc)}{P_{\text{BiasedWalk}}(S|G^qc)} \right)$$

where $\alpha$ is a hyper-parameter

• AVERAGE formulation: Averaging the scores across $n$ walks of $k \epsilon [K]$ lengths.

For a fixed walk length $k$,

$$y_{\theta, \phi}(G_q, G_c, k) = \mathbb{E}_{S \sim P_\phi(S|G^qc)} [y_\theta(G_q, G_c, S)]$$

$$= \sum_{S:|S|=k} y_\theta(G_q, G_c, S) P_\phi(S|G^qc)$$

$$= \sum_{S:|S|=k} y_\theta(G_q, G_c, S) \frac{P_\phi(S|G^qc)}{P_{\text{BiasedWalk}}(S|G^qc)} \ast P_{\text{BiasedWalk}}(S|G^qc)$$

$$= \frac{1}{N} \times \sum_{S \sim \text{BiasedWalk}} y_\theta(G_q, G_c, S) \frac{P_\phi(S|G^qc)}{P_{\text{BiasedWalk}}(S|G^qc)}$$

Finally aggregating over walk lengths,

$$y_{\theta, \phi}(G_q, G_c) = \frac{1}{K} \sum_{k \epsilon [K]} \lambda^k y_{\theta, \phi}(G_q, G_c, k)$$

where $\lambda$ is a hyper-parameter, we kept $\lambda = 1$ unless specified.

5.3 Results

The trainable walks certainly beat the uniform walks in the performance. Though, when compared to other baselines, the trainable walks fall a little short. In the next section we tried to improve upon this gap by taking a new approach of shortest path product graphs.
### Table 5.1: Mean Average Precision values of various approaches. The best performing approaches are emphasised, suggesting superior performance of GxNET. GMN (Graph Matching Networks), RRW (Reweighted Random walk kernel) SPK (shortest path kernel) $\phi$ refers to the trainable parameters of learning transition probabilities in the random walks.

| Dataset         | GxNET (Unbiased walks) | GxNET (Biased walks) fixed $\phi$ | GxNET (Biased walks) trainable $\phi$ | GMN | RRW | SPK |
|-----------------|------------------------|-----------------------------------|--------------------------------------|-----|-----|-----|
| VQA-Clevr (0.10 noise) | 0.98                   | 0.99                             | 0.99                                 | 0.38 | 0.65 | 0.64 |
| SQuAD           | 0.34                   | 0.41                             | 0.42                                 | 0.27 | 0.37 | 0.29 |
| COX2            | 0.31                   | 0.34                             | 0.36                                 | 0.35 | **0.37** | 0.25 |
Chapter 6

Neural matching with Shortest path product graph

6.1 Introduction

After observing the performance of shortest path kernel on SQuAD dataset, we decided to include the idea of shortest path kernel in our product graph setting. The motive behind this was that in a random walk in a product graph of original graphs the maximum hop that the walk can reach is limited while in a product graph of shortest graphs, all nodes are within reach and the information of the distance is also retained. We call this approach Shortest Path Random Walk (SPRW).

6.2 Method

Give a query graph \( G_q \) and corpus graph \( G_c \), we first convert it into its shortest path graphs \( G_{qs} \) and \( G_{cs} \) respectively. Note that the shortest graph is a complete graph with the edges being the distance of the two nodes in the original graphs. Thus evidently, if two nodes are not connected in the original graph then in the shortest graph they will be connected with a edge distance of \( \text{inf} \) (infinity). Further, the tensor product of \( G_{qs} \) and \( G_{cs} \), \( G_{qcs} \) was constructed.

To test out the advantages and the potential of shortest product graphs we experimented with the biased walks by carefully curating the transition probability to first match with the shortest path kernel baseline and then surpass it in performance.

**Start probability.** Instead of a uniform random selection of the start node, we enforced the start nodes to be the matching nodes in the query and corpus graphs, i.e. Give a start node \((u, u')\) in the product graph \( G_{qcs} \), its start probability \( P_{(u, u')} \) will be

\[
P_{(u, u')} = \frac{1 \{ l(u) = l(u') \} + \epsilon 1 \{ l(u) \neq l(u') \}}{\sum_{(u, u') \in G_{qcs}} 1 \{ l(u) = l(u') \} + \epsilon 1 \{ l(u) \neq l(u') \} }
\]

(6.1)

where \( l(u) = \text{label}(u) \) and \( l(u') = \text{label}(u') \) in the original graphs \( G_{qs} \) and \( G_{cs} \)

**Transition probability.** Similarly, transitions to the matching nodes with the same shortest distance in the query and corpus graphs was favoured over other transitions. Moreover, by tweaking the shortest distance limit we could vary our method from shortest path kernel to a more general setting. Formally, for a transition \((u, u') \rightarrow (v, v')\) the transition probability \( P_{(u, u') \rightarrow (v, v')} \) is

\[
P_{(u, u') \rightarrow (v, v')} = \frac{1 \{ l(u) = l(u') \} 1 \{ l(v) = l(v') \} 1 \{ \text{sp}(u, v) = \text{sp}(u', v') \} 1 \{ \text{sp}(u, v) \leq \alpha \}}{\sum_{(u, u') \rightarrow (v, v') \in G_{qcs}} 1 \{ l(u) = l(u') \} 1 \{ l(v) = l(v') \} 1 \{ \text{sp}(u, v) = \text{sp}(u', v') \} 1 \{ \text{sp}(u, v) \leq \alpha \}}
\]

(6.2)
where \( sp(u, v) \) is the shortest distance between nodes \( u \) and \( v \) in \( G_{qs} \) and \( \alpha \) is a hyper-parameter denoting the maximum distance of shortest path to consider for hopping. Hence, if \( \alpha = \text{inf} \), the above formulation reduces to the shortest path kernel.

Note that there is no training involved in SPRW.

6.3 Results

| Dataset   | Random Prediction | SPK | GMN | SPRW (unbiased) | SPRW (biased) \( sp_{max} = 1 \) | SPRW (biased) \( sp_{max} = \text{inf} \) |
|-----------|------------------|-----|-----|-----------------|-------------------------------|----------------------------------|
| MUTAG     | 0.56             | 0.70| **0.78** | 0.648           | 0.675                         | 0.67                             |
| PTC_MM    | 0.53             | 0.55| **0.57** | 0.54            | 0.54                          | 0.54                             |
| PTC_FM    | 0.52             | 0.55| **0.57** | 0.35            | —                            | —                                |

Table 6.1: Mean Average Precision values. The best performing approaches are emphasised, suggesting the superior performance of GMN. Note that GMN (Graph Matching Networks), RW (Random walk kernel) SPK (shortest path kernel), SPRW (Shortest Path Random Walk) has three settings (a) unbiased random walks (b) biased random walks with \( \alpha = 1 \) and (c) biased random walks with \( \alpha = \text{inf} \)

6.4 Conclusion

From our research we draw following conclusions:

- GxNet (biased walks with trainable \( \phi \)) performs better than GxNet (biased walks with fixed \( \phi \)) which in turn is better than GxNet based on uniform random walks (the original SIGIR paper) [4]

- In many cases normalised Shortest Path Kernel beats the Gxnet approach as was found with SQuAD dataset. Un-normalised Shortest Path Kernel still remains beaten by GxNet approach

- On TUDatasets, Shortest Path Kernel performs similar to Random Prediction denoting that it fails to understand the semantic relations in these dataset
Bibliography

[1] Borgwardt, K. M., and Kriegel, H.-P. Shortest-path kernels on graphs. In Proceedings of the Fifth IEEE International Conference on Data Mining (USA, 2005), ICDM ’05, IEEE Computer Society, p. 74–81.

[2] Borgwardt, K. M., and Kriegel, H. P. Shortest-path kernels on graphs. In Fifth IEEE International Conference on Data Mining (ICDM’05) (2005), pp. 8 pp.–.

[3] Cho, M., Lee, J., and Lee, K. M. Reweighted random walks for graph matching. In Computer Vision – ECCV 2010 (Berlin, Heidelberg, 2010), K. Daniilidis, P. Maragos, and N. Paragios, Eds., Springer Berlin Heidelberg, pp. 492–505.

[4] Goyal, K., Gupta, U., De, A., and Chakrabarti, S. Deep neural matching models for graph retrieval. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (New York, NY, USA, 2020), SIGIR ’20, Association for Computing Machinery, p. 1701–1704.

[5] Li, Y., Gu, C., Dullien, T., Vinyals, O., and Kohli, P. Graph matching networks for learning the similarity of graph structured objects, 2019.

[6] Rajpurkar, P., Jia, R., and Liang, P. Know what you don’t know: Unanswerable questions for squad, 2018.

[7] Siglidis, G., Nikolentzos, G., Limnios, S., Giatsidis, C., Skianis, K., and Vazirgiannis, M. Grakel: A graph kernel library in python, 2020.