Deep Neural Matching Models for Graph Retrieval

Kunal Goyal∗
IIT Bombay

Abir De‡
IIT Bombay

Utkarsh Gupta∗
IIT Bombay

Soumen Chakrabarti
IIT Bombay

ABSTRACT

Graph retrieval from a large corpus of graphs has a wide variety of applications, e.g., sentence retrieval using words and dependency parse trees for question answering, image retrieval using scene graphs, and molecule discovery from a set of existing molecular graphs. In such graph search applications, nodes, edges and associated features bear distinctive physical significance. Therefore, a unified, trainable search model that efficiently returns corpus graphs that are highly relevant to a query graph has immense potential impact. In this paper, we present an effective, feature and structure-aware, end-to-end trainable neural match scoring system for graphs. We achieve this by constructing the product graph between the query and a candidate graph in the corpus, and then conduct a family of random walks on the product graph, which are then aggregated into the match score, using a network whose parameters can be trained. Experiments show the efficacy of our method, compared to competitive baseline approaches.

ACM Reference Format:
Kunal Goyal, Utkarsh Gupta, Abir De, and Soumen Chakrabarti. 2020. Deep Neural Matching Models for Graph Retrieval. In Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR ’20), July 25–30, 2020, Virtual Event, China. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3397271.3401216

1 INTRODUCTION

The graph retrieval problem is to return ‘relevant’ or ‘good’ response graphs from a corpus, given a query graph. It has a wide variety of applications. In passage retrieval [11] for question answering and reading comprehension, passage sentences may be parsed into dependency graphs, with node words represented by word embeddings. The query is likewise parsed, leading to a problem of scoring the level of match between two graphs. In image retrieval [4], each corpus image is turned into a suitable scene graph. The query is also parsed into a scene graph and, the task is to return best-matching corpus graphs.

Graph retrieval requires a trainable (task-adaptive) technique that can score (and thereby rank) candidate graphs based on their relevance to a given query graph. However, in each of the above applications, where the underlying networks bear distinctive physical significance, the notion of relevance is complex, latent and domain dependent. Within the scope of vector-space ranking in IR, hand-crafted functions such as TFIDF and BM25 gave way to learning to rank. In the context of network link prediction, heuristics like common-neighbors and Adamic-Adar led to supervised machine learning. In the same style, we seek task-supervised graph retrieval.

Existing works and their limitations. Graph retrieval has been explored [2, 9, 15, 17], but predominantly using explicit, manually engineered structural similarity measures between a query and a corpus graph. They are unable to exploit any existing relevance information from a labeled dataset. Some approaches involve explicit traversal over query and corpus graphs, seeking approximate alignments, which usually becomes exponentially expensive with query and corpus graph sizes.

To overcome the above limitations, a recent line of work focuses on developing deep learning models [7, 8, 13], that learn a neurally guided similarity score between two graphs from labeled relevance judgments, and thus generalize the classical notion of graph kernels [1, 6, 14]. However, they share some limitations, which preclude their widespread deployment in the context of graph retrieval:

i) Graph kernels quantify structural similarity between graphs, providing highest similarity scores between isomorphic graphs, while in most practical applications like image or passage retrieval, a corpus graph may be labeled as relevant or good to a (smaller) query graph if the query is similar to some (in general, unknown) subgraph of the corpus graph.

ii) Traditional graph kernels rarely address how to combine node and edge attributes and graph structures, whereas in practice, the relevance between a query graph and a ‘good’ response graph needs a suitable combination of graph structure and node and edge features.

iii) In addition to the above two drawbacks, deep graph matching models e.g., GMN [8] compare contextual node embeddings of two graphs, while computing the similarity between them. However, these embeddings rely on a feature aggregation method, thereby losing crucial signals at a finer level of resolution. Hamilton et al. [3] embed a logical query to a vector that can be compared to embeddings of single nodes to score them. Consequently, their method cannot be adapted to graph retrieval from a corpus of graphs.

Present work. Responding to the limitations in prior approaches, we design GxNET, a neural supervised model for graph retrieval, which, given a query graph, ranks candidate graphs based on a
relevance measure that is learned from a labeled training set. In
principle, a relevant corpus graph is likely to contain some sub-
graph which has affinity with the query graph. GxNet searches for
such key subgraphs in the corpus graph, which decides its possible
relevance to a given query.

To elaborate, GxNet first generates the tensor product graph
between a query and a candidate graph, and then computes the
neural edge representation on the product graph, using the train-
able embedding vectors of the underlying components. Such edge
representation vectors encode the affinities between the node pairs
in the product graph; in other words, the affinity scores between
the nodes of the underlying query and corpus graphs. The affinity
scores are aggregated using a family of random walks on the
product graph to give a relevance measure between the underlying
query and the corpus graph. The relevance score and the random
walk that induces the score, aid GxNet to uncover the key subgraph
in a corpus graph. Moreover, the design of GxNet is such that it
can be trained end-to-end by maximizing AUC over a set of queries
with their labeled responses, present in the training data.

In a marked departure from recent deep graph matching mod-
els [8, 16] which depend strongly on comparing node embeddings
between two graphs, GxNet computes edge embeddings, which,
together with the node embeddings, are used to compute the rele-
ance score.

GxNet is parsimonious (fewer parameters) compared to recent
deep graph matching models [8, 16]. It is thus better suited to graph
retrieval tasks involving many comparisons. Our experiments on
synthetic and real data show that GxNet offers significant accuracy
boost beyond several baselines. Our code and data are available at
https://github.com/abir-de/graph-search.

2 PROPOSED GRAPH MATCHING MODEL

In this section, we first give a high-level overview of our proposed
neural model GxNet, beginning with the data it is designed for,
Then, we describe an in-depth analysis of the neural architecture
of our model and finally, we provide the training protocol.

2.1 Query and corpus graphs

We are provided a set of query graphs \( Q = \{ G_q = (V_q, E_q, \mathcal{F}_q, T_q) \} \)
and a set of corpus graphs \( C = \{ G_c = (V_c, E_c, \mathcal{F}_c, T_c) \} \). Each query
graph \( G_q \) is associated with a set of good (relevant) corpus graphs
\[
G_{q+} = \{ G_c \in C | \text{Relevance}(G_q, G_c) = 1 \}
\]  
and a set of bad (irrelevant) corpus graphs
\[
G_{q-} = \{ G_c \in C | \text{Relevance}(G_q, G_c) = 0 \}.
\]  
For each query or corpus graph \( G_q \) we also observe a set of node
features \( \mathcal{F}_q \) and a set of edge types \( \mathcal{T}_q \), where \( \bullet \) may be one of \( q \) and \( c \). The meaning of the underlying node features
and edge types vary across applications. E.g., consider an image
retrieval task, where the images show the positions of different shapes
of objects. Then we may have \( \mathcal{F}_q \) with \( \{ \text{white ball}, \text{white cube}, \text{black ball}, \text{black cube} \} \) and \( \mathcal{T}_q \) with \( \{ \text{front}, \text{behind}, \text{back}, \text{right} \} \). Dependency graphs can be captured within the same formalism.

2.2 High level overview

GxNet takes the set of query graphs \( Q \) and the set of corpus graphs
\( C \) as inputs and associates each pair of query and corpus graphs
\( (G_q, G_c) \) with a score \( y(G_q, G_c) \) as output, which measures the
confidence that \( G_c \) is a relevant graph to \( G_q \), i.e., \( G_c \in G_{q+} \). To do so,
first builds a tensor product \( G^{\text{TF}} \) between \( G_q \) and \( G_c \) and then,
neurally computes the affinity scores for each pair of nodes in the
product graph \( G^{\text{TF}} \); or equivalently, the affinity scores between
each node pair of \( G_q \) and each node pair of \( G_c \). If \( G_c \) is relevant to
\( G_q \), then (with a high likelihood) there exists a key subgraph in
\( G_c \) which has a high affinity with \( G_q \). In that case, \( G^{\text{TF}} \) would also
contain a subgraph — induced by \( G_q \) and the latent key subgraph of
\( G_c \) — that has a high value of affinity score. In order to compute such
a subgraph in \( G^{\text{TF}} \), we first aggregate the affinity scores of different
subgraphs, sampled using several random walks with restart,
and then aim to extract the subgraph with highest affinity score. Next,
we detail the neural architecture of GxNet.

2.3 Neural modeling of GxNet

Given \( G_q \in Q \) and \( G_c \in C \), we use a neural network over the
product graph, as constructed below.

Product graph. For each pair of query graph \( G_q \) and corpus graph \( G_c \),
we construct the tensor product graph \( G^{\text{TF}} = G_q \times G_c \) [12]:
\[
G^{\text{TF}} = G_q \odot G_c = (V_q \times V_c, E_q \times E_c, F_q \times F_c, T_q \times T_c)
\]  
where
\[
V^{\text{TF}} = V_q \times V_c, E^{\text{TF}} = E_q \times E_c, F^{\text{TF}} = F_q \times F_c, T^{\text{TF}} = T_q \times T_c.
\]  
Edges are included in the product graph \( G^{\text{TF}} \) according to the rule
\((u, u') \in E^{\text{TF}} \iff (u, u') \in E_q \quad \text{and} \quad (u', u') \in E_c\); (4)
the node feature for a node \((u, u') \in G^{\text{TF}} \) are set as follows:
\[
f(u, u') = (f_u, f_{u'}) \quad \forall \in G_q, u' \in G_c;
\]  
and the edge labels in \( G^{\text{TF}} \) are set as follows:
\[
t(u, u') = t_{u, u'} \quad \text{if} \quad (u, v) \in E_q \quad \text{and} \quad (u', v') \in E_c.
\]  

Embeddings in the product graph. Next we design a neural
model of GxNet, which takes a query graph \( G_q \) and a corpus graph
\( G_c \) as inputs, and computes \( y_q(G_q, G_c) \): the confidence score
for relevance of \( G_c \) to the query \( G_q \). Here \( \theta \) represents the trainable
parameters of the neural networks. To do so, GxNet first turns
the node features \( f_u \) and edge types \( t_{u, u'} \) into suitable vectors \( f_u \) and \( t_{u, u'} \).
The exact conversion protocol may vary across different
tasks. For example, we can simply compute \( f_u \) and \( t_{u, u'} \) as the
one-hot encoding of \( f_u \) and \( t_{u, u'} \) respectively. Then, from individual node
\( f_u \) and \( t_{u, u'} \) in constituent graphs \( G_q \) and \( G_c \), GxNet computes an embedding \( z_{u, u'} \). For each node \((u, u') \) in the product graph \( G^{\text{TF}} \). In addition, GxNet also computes an embedding \( r_{u, u'} \) for each edge in the
product graph based on the base edge embeddings \( r_{u, u'} \). More
specifically, we have:
\[
z_{u, u'} = g_{\text{node}}(f_u, f_{u'})
\]  
\[
r_{u, u'} = g_{\text{edge}}(t_{u, u'}, t_{u', u'})
\]  
In the next step, GxNet constructs the embedding vector for each
source (node, edge, target node) triple in the product graph \( G^{\text{TF}} \),
denoted
\[
\tau = ((u, u') \rightarrow (v, v'))
\]  
\[
as \quad h_{u} = \rho(z_{u, u'}, r_{u, u'}, z_{v, v'}); \quad \text{and}
\]  
\[
s = \sigma(W_{s}h_{u}),
\]  
which in turn gives us a score or scalar weight of \( \tau \), given by

1702
where $s_r$ indicates an affinity score for the edge $r$. Here $g_{\text{node}}$, $g_{\text{edge}}$, $p$, $\sigma$ are suitable standard networks with appropriate non-linearities. We let $\theta$ stand for the set of all the trainable parameters in these networks. The exact choices of these non-linearities are described in detail in the following section.

**Computing $y_\theta(G_q, G_c)$ using neural score aggregation.** If $G_c$ is a ‘good’ (relevant) graph for $G_q$, then, with a high likelihood, it contains a subgraph that has high affinity to $G_q$. Such a subgraph of $G_c$ also induces a subgraph $S$ in $G^\text{PP}$, that has a high value of aggregated score $\sum_{r \in S} s_r$. All in, we need to identify this subgraph $S$ and the aggregated score $\sum_{r \in S} s_r$. Computing such a subgraph in general is intractable. Therefore, we resort to a Monte Carlo sampling approach, where we sample subgraphs using several random walks on the product graph $G^\text{PP}$, and choose the one with maximum score.

We first fix a hyperparameter $K$, the maximum length of the random walks. For each $k \in [K]$, we perform $n$ random walks with restarts in the product graph. The start probability of all nodes are equal. At any node, transition probabilities for all edges are equal. After sampling a random walk, we sample the subgraph as the induced subgraph of random walk. After sampling subgraphs $S_i$, $i \leq n$. Then we compute the approximate maximum of the aggregated affinity scores in the following way.

$$\text{score}_k(G_q, G_c) = \sum_{i \in [n]} \sum_{r \in S_i} \exp\left(\sum_{s \in S_i} ||S_i|| = k \cdot s_r \right) \sum_{r \leq n} \exp\left(\sum_{s \in S_i} ||S_i|| = k \cdot s_r \right)$$

Finally, we compute the average affinity over the different lengths of the walk, penalized by the length of path $k$. Hence, we have the final confidence score

$$y_\theta(G_q, G_c) = \frac{1}{K} \sum_{k \in [K]} \frac{\text{score}_k(G_q, G_c)}{k},$$

which indicates the confidence that $G_q$ is relevant to $G_c$.

**Model inference.** Given the set of query graphs $Q$ and the set of good and bad corpus graphs $G^+_q$ and $G^-_q$, for each query $G_q \in Q$, we estimate the parameters $\theta$ of GxNet by minimizing the pairwise good-bad loss, i.e.,

$$\argmin_{\theta} \sum_{q \in Q} \sum_{G^+_q, G^-_q} \text{ReLU}\left(\Delta + y_\theta(G_q, G^+_q) - y_\theta(G_q, G^-_q)\right)$$

where $\Delta$ is a tuned margin and ReLU is max{1, 0}.

### 3 EXPERIMENTS

In our implementation, for the VQA task, we consider $g_{\text{node}}$ and $g_{\text{edge}}$ to be an MLP with two layers 5 nodes each; $p$ to be an MLP with two layers 10 nodes each; and $\sigma$ to be sigmoid function. Moreover, we set $\dim(z)$, $\dim(r)$ and $\dim(p)$ as 5, 5 and 10 respectively. For SQuAD data set we consider $g_{\text{node}}$ and $p$ to be an MLP with two layers 32 nodes each; $g_{\text{edge}}$ to be an MLP with two layers 8 nodes each. dim($z$), dim($r$) and dim($p$) as 32, 8 and 32 respectively.

**Evaluation protocol.** Given a set of queries $Q$ and the set of good and bad corpus graphs $G^+_q$ and $G^-_q$ for each query $q \in Q$, we randomly split both $G^+_q$ and $G^-_q$ in 80% training and 20% test set. Then we train GxNet on the training set and based on the trained model $\theta$, we rank the corpus graphs in the test set in the decreasing order of the confidence score $y_\theta$ (Eq. 13). Then we evaluate the ranked list using three well known measures of accuracy [10]: precision-at-rank-$k$, mean average precision (MAP), and mean reciprocal rank (MRR).

**Baselines.** We compare GxNet with two baselines: (i) Re-weighted Random Walk Graph Matching (RRWM) [2], which is a simple iterative graph matching method that does use any learning method and (ii) Graph matching network (GMN) [8], which is a deep learning based method. Since RRWM does not involve any learning, we only run it over the test set.

**VQA data preparation.** The Clevr VQA benchmark [5] consists of an image corpus where each image is associated with a scene graph. The nodes of a scene graph have attributes corresponding 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. Here we directly generate query graphs.

The goal here is to test the resilience of graph retrieval systems to structural distortions between query graphs and (approximately matching) subgraphs of corpus graphs. We first generate query graph nodes 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 ($\text{size} \in \mathbb{R}$) node attributes. Each component is in $[0, 1]$, to which we add an independent Gaussian noise with standard deviation $\sigma$ which we vary, and clip the attribute values back to $[0, 1]$ if needed.

**VQA results and discussion.** Figure 1(a) compares Precision@$K$ between GxNet, RRWM and GMN for the VQA task. It shows that the performance of GMN is particularly poor, despite training a deep neural model. This was surprising to us. A careful investigation reveals that, at a high level, GMN matches the node representations between two graphs. However, traditional node embeddings depend strongly on a feature aggregation protocol, which often weakens crucial structural signals. The performance of RRWM is better than GMN. Unlike GMN, RRWM does not leverage the presence of training labels in the datasets. However, its underlying
graph traversing method enables it to compare two graphs at a more granular level than GMN. GxNet outperforms both GMN and RRWM. As Figure 2 shows, all test queries show better AP and RR with GxNet than each of GMN and RRWM for non-noisy VQA data. Figure 3 illustrates the effect of noise. While GxNet remains superior, reducing susceptibility of GxNet to noise is an interesting avenue for future work.

Figure 2: AP and RR gains of GxNet on VQA (no attribute noise) task. Queries Q sorted by increasing gain along x-axis.

Figure 3: Effect of introducing noise to node attributes in VQA query and corpus graphs on MAP and MRR.

Figure 4: MRR (=MAP, given one relevant item per query) losses/gains of GxNet vs. RRWM and GMN on SQuAD. Queries Q sorted by increasing gain along x-axis.

SQuAD data preparation and results. In the SQuAD text QA benchmark [11], an instance consists of a query, a passage, and a gold answer span marked in the passage. Dependency parse trees can be obtained (https://spacy.io/) from each query and sentence. Similarity between the query dependency graph and a sentence dependency graph is a useful feature for ranking sentences and passages. We wish to learn this similarity to improve the rank of the graph corresponding to the sentence with the gold answer span. For evaluation we considered only those query graphs with 4–10 nodes and corpus graphs with 10–20 nodes. The number of queries sampled for training, testing and validation are 1000, 100, 100 respectively. For each query, 10 corpus graphs were sampled, one being relevant and others irrelevant. Train, test and validation query graphs have 8.43, 8.0, 8.4 nodes on average. For corpus graphs the corresponding numbers are 15.9, 16.3, and 15.8. Figure 1(b) compares Precision@K between GxNet, RRWM and GMN for SQuAD. The MAP of GxNet is 0.455, while that of GMN and RRWM are 0.273 and 0.406 respectively. The MAP of GMN is close to MAP under random ranking. Figure 4 shows query-by-query losses and gains; overall, GxNet continues to lead.

4 CONCLUSIONS

We presented GxNet, a trainable neural system for retrieving from a corpus of graphs those that best match a given query graph. GxNet creates a product graph between the query and a candidate graph in the corpus, conducts random walks in the product graph, and computes a suitable trainable aggregation of certain properties of these walks. Training is done via supervised relevant and irrelevant graphs for a set of queries. Preliminary experiments show that GxNet performs substantially better than recent approaches.

Acknowledgment. Partly supported by Amazon, IBM and nVidia.

REFERENCES

[1] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. Vishwanathan, A. J. Smola, and H.-P. Kriegel. Protein function prediction via graph kernels. Bioinformatics, 21 (suppl_1):i47–i56, 2005.
[2] M. Cho, J. Lee, and K. M. Lee. Reweighted random walks for graph matching. In ECCV, 2010.
[3] W. Hamilton, P. Bajaj, M. Zitnik, D. Jurafsky, and J. Leskovec. Embedding logical queries on knowledge graphs. In NeurIPS, 2018.
[4] J. Johnson, R. Krishna, M. Stark, L.-J. Li, D. Shamma, M. Bernstein, and L. Fei-Fei. Image retrieval using scene graphs. In CVPR, 2015.
[5] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. Lawrence Zitnick, and R. Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In IEEE CVPR, pages 2901–2910, 2017.
[6] R. Kondor and H. Pan. The multiscale laplacian graph kernel. In Advances in Neural Information Processing Systems, pages 2990–2998, 2016.
[7] T. Lei, W. Jin, R. Barzilay, and T. Jaakkola. Deriving neural architectures from sequence and graph kernels. In ICML, 2017.
[8] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli. Graph matching networks for learning the similarity of graph structured objects. In ICLR, 2019.
[9] Y. Liang and Z. Zhao. Similarity search in knowledge graphs: A multi-layered indexing approach. In ICDE, pages 783–794. IEEE, 2017.
[10] C. D. Manning, P. Raghavan, and H. Schütze. Introduction to information retrieval. Cambridge university press, 2008.
[11] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. arXiv preprint arXiv:1606.05255, 2016.
[12] E. Sampathkumar. On tensor product graphs. Journal of the Australian Mathematical Society, 20(3):268–273, 1975.
[13] M. Togninalli, E. Ghisu, F. Llinares-López, B. Rieck, and K. Borgwardt. Wasserstein weisfeiler-lehman graph kernels. NeurIPS, 2019.
[14] S. V. N. Vishwanathan, N. N. Schraudolph, R. Kondor, and K. M. Borgwardt. Graph kernels. Journal of Machine Learning Research, 11(Apr):1201–1242, 2010.
[15] X. Yan, F. S. Yu, and J. Han. Substructure similarity search in graph databases. In SIGMOD, 2005.
[16] T. Yu, R. Wang, J. Yan, and B. Li. Learning deep graph matching with channel-independent embedding and hungarian attention. In ICLR, 2020.
[17] Y. Zhu, L. Qin, J. X. Yu, and H. Cheng. Finding top-k similar graphs in graph databases. In EDBT, 2012.